NEC Electronics Inc.

NEC

USER'S MANUAL

December 1985

μPD7220/7220A

Graphic Display Controller

NEC Electronics Inc.

NEC

μPD7220/7220A

Graphic Display Controller

December 1985
Stock No. 500230

CONTENTS

CONTENTS (cont)

The μPD7220 Graphics Display Controller (GDC) is an intelligent microprocessor peripheral designed to be the heart of a high-performance raster-scan computer graphics and character display system. Positioned between the video display memory and the microprocessor bus, the GDC performs the tasks needed to generate the raster display and manage the display memory.

Processor software overhead is minimized by the GDC's sophisticated instruction set, graphics figure drawing, and DMA transfer capabilities. The display memory supported by the GDC can be configured in any number of formats and sizes up to 256K 16-bit words.

The display can be zoomed and panned, while partitioned screen areas can be independently scrolled. With its light pen input and multiple controller capability, the GDC is ideal for advanced computer graphics applications.

Features

- ☐ Microprocessor interface
  - DMA transfers with 8257- or 8237-type controllers
  - FIFO command buffering
- ☐ Display memory interface
  - Up to 256K words of 16 bits
  - Read-Modify-Write (RMW) display memory cycles in under 734 ns
  - Dynamic RAM refresh cycles for nonaccessed memory
- ☐ Light pen input
- ☐ External video synchronization mode
- ☐ Graphics mode
  - Four megabit, bit-mapped display memory
- ☐ Character mode
  - 8K character code and attributes display memory
- ☐ Mixed graphics and character mode
  - 64K if all characters
  - 1 megapixel if all graphics
- ☐ Graphics capabilities
  - Figure drawing of lines, arc/circles, rectangles, and graphics characters in 734 ns per pixel
  - Display 1024-by-1024 pixels with four planes of color or grayscale
  - Two independently scrollable areas
- ☐ Character capabilities
  - Auto cursor advance
  - Four independently scrollable areas
  - Programmable cursor height
  - Characters per row: up to 256
  - Character rows per screen: up to 100
- ☐ Video display format
  - Zoom magnification factors of 1 to 16
  - Panning
  - Command-settable video raster parameters
- ☐ Single +5 volt, NMOS, 40-pin DIP technology
- ☐ DMA capability
  - Byte or word transfers
  - 4 Clock periods per byte transferred in word mode
  - 5 Clock periods per byte transferred in byte mode

System Considerations

The GDC is designed to work with a general purpose microprocessor to implement a high-performance computer graphics system. Through the division of labor established by the GDC's design, each of the system components is used to the maximum extent through a six-level hierarchy of simultaneous tasks.

At the lowest level, the GDC generates the basic video raster timing, including sync and blanking signals. Partitioned areas on the screen and zooming are also accomplished at this level. At the next level, video display memory is modified during the figure drawing operations and data moves. Third, display memory addresses are calculated pixel by pixel as drawing progresses. Outside the GDC at the next level, preliminary calculations are done to prepare drawing parameters.

At the fifth level, the picture must be represented as a list of graphics figures drawable by the GDC. Finally, this representation must be manipulated, stored, and communicated. By handling the first three levels, the GDC takes care of the high-speed and repetitive tasks required to implement a graphics system

INTRODUCTION

The μPD7220 Graphics Display Controller (GDC) interfaces to its display memory using 20 of its 40 pins. The GDC assumes full responsibility for controlling all aspects of this interface, including generating the raster-scan address sequence, the reading and modification of display memory data, and coordination of these activities. This interface, as described below, controls a fully functional video system, but extra hardware can be added to provide more advanced systems with higher performance levels.

The 20 interfacing lines are used for several purposes. The three operating modes of the GDC vary the use of some of the pins, as will be noted later. Graphics mode pin usage will be described first. Sixteen pins are used for the bidirectional address and data bus, AD_0 to AD_{15}, which forms the heart of the interface. Two lines, A_{16} and A_{17}, output the top two bits of the display memory address and provide system control signals in operating modes other than graphics mode. The last two lines, ALE (or $\overline{RAS}$) and $\overline{DBIN}$, provide the timing and control information necessary to coordinate external hardware with the GDC's bus timing.

In addition to these 20 lines, the blanking (BLANK) and horizontal sync (HSYNC) outputs are also involved in the display memory interface. The assertion of BLANK is coordinated with the bus cycle timing and type so that only active display information reaches the CRT screen. Horizontal sync is used in one of the display modes to key the multiplexing of system control information over the A_{16} and A_{17} pins during horizontal retrace blanking.

Although the display memory interface provides addresses during the display raster-scanning process, it is not involved in the video pixel rate operations. The frequencies involved are often too high for any MOS LSI device. Therefore, external hardware must handle the parallel-to-serial conversion of display memory words into pixels, etc. Although the GDC supports zoomed display magnification and 32-bit wide display cycle accesses (twice the normal 16-pixel width), external hardware must be designed to support these capabilities. For capabilities beyond these basic ones, additional external hardware can be used to add many features such as dual ported memory, smooth horizontal panning, and video look-up tables. This partitioning of functions between the GDC and external hardware makes possible a great deal of design flexibility.

CLOCKING

A single clock drives the GDC and all the internal logic of the IC and is used as the basis for display timing generation. This TTL-compatible input runs at twice the memory display cycle rate. Therefore, with a 5-MHz clock, normal display cycles will take 400 ns (two clock cycles of 200 ns each). During this time, either 16 or 32 bits will be accessed from video display memory and loaded into the parallel-to-serial video shift registers. The clock input pin of the GDC is labeled 2xWCLK to emphasize that two clock cycles are used for each nonzoomed display memory cycle. Note that the frequency of this clock signal will be determined by the requirements of the video display CRT unit. A low-speed CRT, for example, will need video data generated at a rate below the maximum.

Other memory cycle types use different numbers of clock cycles but are always a multiple of two clock periods. This fundamental time period will be referred to as a "word time." Read-modify-write memory cycles use four cycles of the 2xWCLK when the display zoom factor is one or two. For zoom factors greater than two, RMW cycles are extended to the same length as the lengthened zoomed display cycles. Zoom-magnified display cycles use two extra clock periods per zoom factor increment.

The 2xWCLK input is easy to drive with a TTL signal. Voltage levels required for the clock input are listed below.

Low level: −0.5 V to 0.6 V
High level: 3.5 V to V_{CC} + 1.0 V

In order to ensure a good noise immunity margin in the high state, a pull-up resistor can be used to achieve voltages in excess of 3.5 V, and the driving TTL output can be lightly loaded to provide extra low-state noise immunity. All other GDC inputs require the following logic levels.

Low level: −0.5 V to 0.8 V
High level: 2.2 V to V_{CC} + 0.5 V

BUS CONTROL SIGNALS

Address Latch Enable ALE [$\overline{RAS}$]

The ALE signal identifies the start of a memory cycle. This is important because the 2xWCLK clock is running at twice the fastest memory cycle rate, making it impossible for external circuitry to predict which clock cycle will be the first one of the upcoming memory cycle. Also, the differing lengths of blanked, active display, zoomed display, and read-modify-write (RMW) memory cycles demand a constantly adapting determination of the first clock cycle. Since ALE always marks the start of a new memory cycle, it must be used to synchronize the external circuitry to the GDC's bus timing.

The falling edge of ALE indicates the first clock cycle of a new memory cycle and the availability of the memory address on the AD_0 through AD_{15} pins and A_{16} and A_{17}. External circuitry should begin the row and column address strobing of dynamic RAMs after the occurrence of this edge. The precharge time required by the RAMs can often be provided by the ALE high time which precedes this falling edge. Indeed, the timing of this signal may be used as the $\overline{RAS}$ strobe, fed directly to the display memory RAMs. In any case, the required external hardware will be simple, but the detailed implementation will be application dependent.

For example, in high speed systems it may be impossible to finish the $\overline{RAS}$ and $\overline{CAS}$ sequence before the address supplied by the GDC begins to disappear (in preparation for data input during RMW cycles). In this case it will be necessary to latch the column address component to ensure that data hold times are met at the RAMs, relative to the strobes. Static RAM display memories will have different timing requirements than dynamic RAM designs.

The ALE signal must also be used to control the loading of the video shift registers. External hardware should load the shift registers with the data accessed from the RAMs at the end of the second clock cycle. This can be allowed to happen regardless of memory cycle type, since the GDC will assert its blanking output pin to suppress any nonactive display access such as RMW, and retrace cycles.

Data Bus Input Enable $\overline{DBIN}$

The assertion of the $\overline{DBIN}$ output identifies the time the GDC will accept the data read from the RAMs during RMW cycles. The GDC will input the data found at its AD lines near the end of this $\overline{DBIN}$ low period and use it during the modify operation to generate the data which will be written back into the originally specified memory location late in the RMW cycle. The resulting data is output without further indication by the GDC, so that external logic must generate the necessary write pulse and any bus turn-around control signals. These signals can easily be generated using the $\overline{DBIN}$ signal from the GDC. How the $\overline{DBIN}$ signal is used to generate signals depends on the particular system architecture under consideration. The $\overline{DBIN}$ assertion is the only direct way to differentiate an RMW cycle from other memory cycles.

MULTIPLEXED ADDRESS AND DATA BUS

AD_0 through AD_{15}

The sixteen pins AD_0 to AD_{15} form the main pathway between the GDC and the video display memory for both address and data information. During every memory cycle, regardless of type, the low 16 bits (13 bits in character mode) of an address in the display memory are sent out over these pins. Together with the A_{16} and A_{17} output-only pins, up to an 18-bit address is provided. During read-modify-write (RMW) cycles, the 16-bit data read from the display memory is then input over these lines to the GDC in response to the $\overline{DBIN}$ signal. Later in the RMW cycle, the resulting data is output back to the display memory over this same pathway.

The AD_{13} to AD_{15} pins are used differently when the GDC is operated in the "coded characters only" (character) mode. The least significant three bits of the GDC's internal line counter are output instead of the top three bits of address and data information. The lower thirteen AD lines are used as described above, thereby limiting the display memory to 8K 13-bit words. In coded character display applications this is many pages of text and many attribute bits. By making direct use of the GDC's internal line counter to drive the character generator, the external hardware can be very simple.

A_{16} and A_{17}

These two output-only pins are used differently in each of the GDC's operating modes. In graphics mode, they are used in conjunction with the AD_0 to AD_{15} pins to provide an 18-bit address in display memory. In character mode, A_{17} outputs the cursor signal while A_{16} supplies the most significant bit of the line counter. In mixed mode, they are multiplexed so that during horizontal sync pulses, A_{17} signals whether the upcoming raster line should be interpreted as bit-mapped graphics or characters, and A_{16} outputs the external line counter clear pulse. During the active display time, A_{17} indicates the cursor position (in a character area), and A_{16} follows the attribute blinking timing.

The values at these pins can change during the first clock cycle of the memory cycle and the last two clock cycles of an RMW cycle. During mixed mode operation these values are available ten clock cycles after the trailing edge of the horizontal sync (HSYNC) period.

In graphics mode, the A_{16} and A_{17} address values become stable shortly after the start of the memory cycle, and stay stable until the beginning of the third clock cycle of an RMW cycle. If they are used for decoding the address of a write pulse going to a specific group of memory chips, they must be latched with an ALE signal at the beginning of the RMW cycle. They should be used with latching to drive the select inputs of the decoders which are used to steer the $\overline{DBIN}$ and $\overline{WE}$ pulses to the display memory system. In multiplane systems, A_{16} and A_{17} can be used for selection of one of four planes during an RMW cycle.

ADDITIONAL SIGNALS

Horizontal Sync

The horizontal sync (HSYNC) output indicates the time during the raster-scanning sequence when the CRT's beam is to start its retrace back to the left side of the screen. The GDC provides a very flexible programming format to allow both its position and width to be specified to one word time (which is the same as two clock cycles).

The memory cycles during HSYNC pulses are used by the GDC to output the dynamic RAM refresh addresses over the lower eight AD lines. This function can be enabled or disabled by the RESET command. The refresh address is supplied from the GDC's internal 8-bit refresh counter. Since these memory cycles are differentiated from all others by occurring during HSYNC, RAS-only refresh cycles can be easily generated for all RAMs with a few gates. This is important for systems in which the RAMs for one plane have their outputs tied together to drive individual bit lines (that is, thirty two 16K RAMs form a 32K by 16-bit plane). Some systems will be designed with only one group of RAMs per plane (one RAM per bit) and therefore will not have outputs tied together (that is sixteen 64K RAMs form a 64K by 16-bit plane). For these systems there is no need to suppress $\overline{CAS}$ and force $\overline{RAS}$ for all RAMs during HSYNC periods, since there will be no bus contention and all RAMs will already be getting every $\overline{RAS}$.

Blanking

The blanking output from the GDC signals when an active display cycle is occurring so that external hardware can pass the information accessed from display memory to the CRT display. Any other type of memory cycle should be blanked before the video information reaches the CRT.

In either coded-character or bit-mapped graphics applications, the data word fetched from memory will be loaded into a register as the first step in the generation of the serial video stream that is eventually sent to the CRT display unit. Often this process will take one or more display pixels, depending on the design.

For graphics, this first register is often a shift register which does a parallel-to-serial conversion to produce the video serial pixel stream. Further pipeline processing hardware can be added for gray-scale capability and look-up-tables, for example. For coded characters, the first register often holds the data while a character generator is accessed. A shift register then converts the character generators' output from parallel to serial form, which in turn feeds the attribute logic.

Of course, along each of these video pipelines the video data is held in registers which introduce synchronous delays. The blanking signal from the GDC must be kept aligned with the data in the video pipeline until it is ANDed with the video data, so that only valid data reaches the display screen. This is easily accomplished if the blanking signal experiences the same synchronous delays as the video data before the point where the aligned blanking signal is combined with the video data to allow it to force black on the screen during blanked intervals.

Given synchronously aligned blanking, there is no reason to suppress nonactive display cycle loads of the video pipeline since they will automatically be blanked by the GDC anyway. This feature contributes to a simple interface between the GDC and the display memory.

VIDEO DISPLAY MEMORY CYCLES

The main role of the GDC is to manage the video display memory during both the modification of data in the memory and the raster-scanning process. By giving the responsibility for both of these competing processes to the GDC, memory cycle allocation can be readily handled. The determination of the next memory cycle type is done by the GDC via a decision process that includes inputs from the host microprocessor at initialization time and internal status including the video sync generator and the command processor. The result is a smoothly operating display system which provides the user with a great amount of flexibility in configuring his or her system, and with effective utilization of the display memory's bandwidth.

For the first time in a single LSI device, management of these two competing tasks, data modification and video display, can be accomplished automatically with no external hardware. In the past this arbitration and switching process required a dual-ported display memory or DMA hardware. Although these options are still open to the GDC system designer, they are no longer the only choices. The GDC will modify display memory at any time during the raster-scanning process for maximum writing speed, or it can be configured by software to do the modify operations only during the horizontal and vertical flyback times. By not doing RMW cycles during the active display portion of the raster, no disturbances are visible on the screen. The price paid for this is a lower drawing speed.

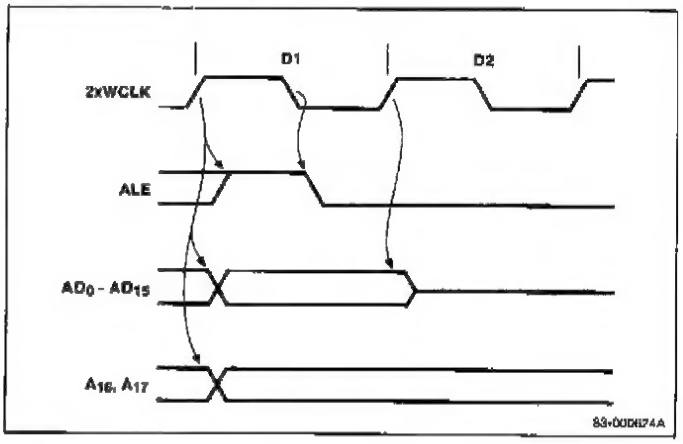
For those who want high-speed drawing with no display disturbances, a dual-ported display memory and two GDCs can be used. The first GDC does nothing but raster-scanning display cycles, while the second GDC does all the drawing. External TTL hardware then interleaves their accesses into the display memory. The two GDCs must use the same clock (or a multiple of the same clock) so they will stay cycle-aligned, and the display memory must have a bandwidth high enough to support both GDCs' accesses without contention.

Basic Memory Cycle Timing

All GDC timing is referenced to the 2xWCLK clock which drives the GDC. Two clock cycles are required to complete one machine cycle within the GDC. This time period therefore represents the minimum time quantum for any bus cycle type. Therefore every memory cycle will use an even number of clock cycles, which we will number starting with 1. The rising edge of 2xWCLK signals the start of each cycle.

Every memory cycle starts by outputting a display memory address over the AD_0 through AD_{15} lines, and the A_{16} and A_{17} lines. For the first half of the first cycle, the ALE output is high. Its falling edge just after the middle of the first cycle signals the availability of this address information. It is important for external hardware to use this edge and not the level of ALE to signal the start of the memory cycle because ALE may assume a high level long before the first cycle begins during zoomed display operation. The minimum high time for the ALE output is one-half of a clock cycle (the first half of the first cycle), adjusted for possibly differing delay times of the two edges of 2xWCLK to the ALE output. The address lines begin to output the address value with the start of the first cycle, and hold it steady until after the end of the cycle. The A_{16} and A_{17} outputs can change value with the start of the first cycle and will be stable until the fourth cycle of an RMW operation or the next memory cycle begins. Figure 2-1 shows these relationships.

Figure 2-1. Outputting the Display Memory Address



Note that the AD lines return to the high-impedance state at the end of the first clock cycle regardless of the memory cycle type. At this point there is no way to tell explicitly if this memory cycle is a display cycle, a read-modify-write cycle, a zoomed display cycle, or a blanked retrace cycle. In fact, until the middle of the second cycle, neither ALE or $\overline{DBIN}$ gives any indication of the type of cycle that is in progress. There is normally no need to know the cycle type at this time in the cycle. Regardless of the cycle type, the addresses generated by the GDC are directed to the display memory.

Dynamic Memory Timing Signals

The larger display memories used for bit-mapped graphics encourage the use of low-cost dynamic memory. Although somewhat more complex to control than static memory, the cost-per-bit savings of dynamic memory quickly offset the cost of the extra TTL-drive hardware. The economics of a display memory as large as the one the GDC can control (512K bytes) require the use of dynamic RAMs. With this reality in mind, the GDC was designed to make this interface as easy as possible.

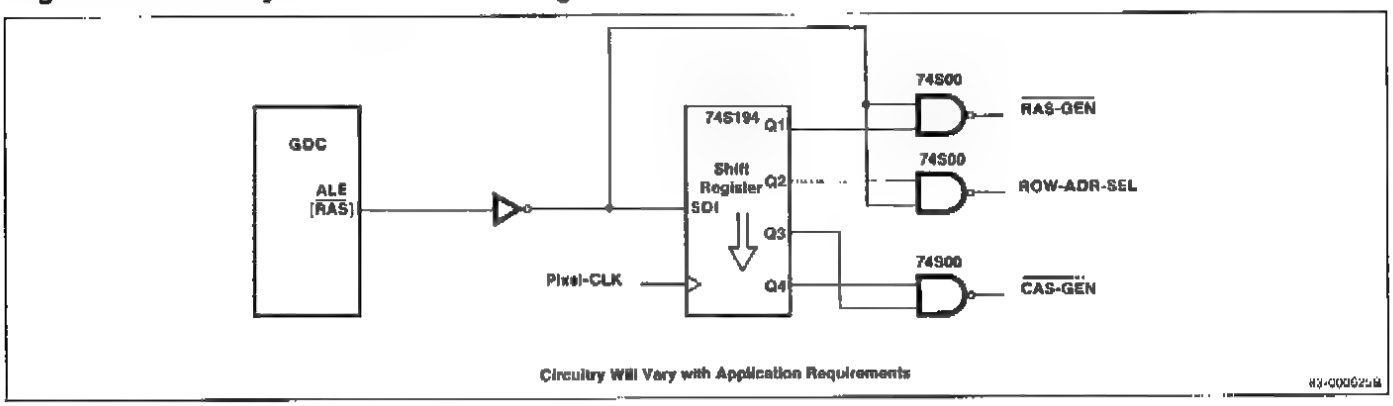
The first requirement when driving dynamic RAMs is to strobe the row and column addresses into the RAMs early in the memory cycle. See figure 2-2. The GDC indicates the time to start this process with the falling edge of the ALE ($\overline{RAS}$) output. The RAMs can then be $\overline{RAS}$-strobed with the row address at their address inputs, the address switched to the column address, and the $\overline{CAS}$ strobe supplied. The GDC plays no direct part in this process except when the ALE ($\overline{RAS}$) output can be used directly as the $\overline{RAS}$ strobe. (This will be possible if the ALE high time during the first half of the first clock cycle is long enough to satisfy the RAM precharge time requirement.)

These timing signals can be generated with a few TTL packages by using the high-frequency oscillator signal, which is running eight times faster than a 2xWCLK clock cycle, for example. More than likely, there is enough timing resolution using the pixel clock to drive a digital delay shift register which is following the GDC's ALE output. Simple gating can then be used to generate the $\overline{RAS}$ and $\overline{CAS}$ strobes and the address select control signal.

Looking at the overall framework these signals must work within, the main constraint is to have the output data from the RAMs available and set up at the video shift register's inputs at the end of the second clock cycle. The timing requirement for RMW cycle read data is more relaxed than for display cycles and requires no extra hardware when the $\overline{DBIN}$ output timing can be used directly. During RMW cycle write periods, the WE pulse can often be generated from 2xWCLK timing without any pixel rate clocking. Each display system design must be evaluated individually to ensure that all the timing requirements are met before the particular timing generator can be finalized. The tremendously wide range of applications suitable for the GDC ensures that no one design will cover all situations.

The second requirement when using dynamic RAMs is to make sure that all row addresses within each RAM are accessed during each RAM refresh period. In some cases where the display is not zoomed and large area DMA accesses are not made in the display memory, the raster-scanning process itself will be sufficient to maintain dynamic RAM refresh. In many applications, however, the raster-scanning process will be interrupted for too long (by figure drawing, DMA transfers, or zoomed display) to keep the RAMs refreshed. For these situations, the GDC has an 8-bit counter that can generate successive refresh addresses on the least significant AD lines during HSYNC time, if so programmed. Using this facility, an arbitrarily complex display memory structure can be implemented using multiple RAM pages and planes and still maintain dynamic RAM refresh for all the devices.

Figure 2-2. Memory Strobe Generation Logic



Display memory can be organized in a number of ways. When using "by one" RAMs such as 16K-by-1 or 64K-by-1 devices, the most common minimum group of RAMs is 16 devices with each RAM storing one bit of the GDC's 16-bit word. (It should be noted that smaller word sizes may be used with the addition of a few programming constraints. Further discussion of this option is provided in a later section.) These groups can be organized so that, first, they share a RAM data output bus to form a bigger linear range of addresses, and second, their outputs can be kept separate and used with independent bus buffers and video shift registers. The first technique, paging, is used to make a bigger display memory than one group of RAMs can provide, while the second technique, using planes, provides more bits of information at each pixel position. Both techniques can be combined in the same system.

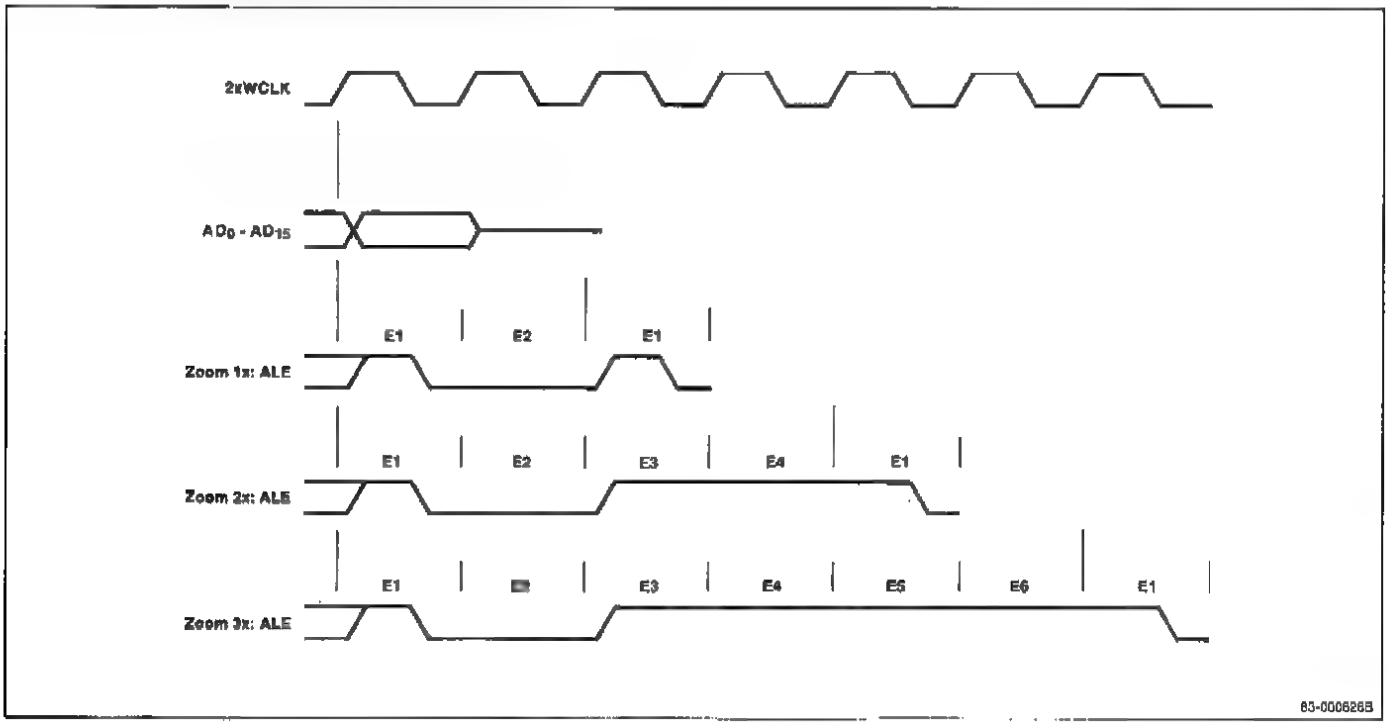
The approach used to guarantee that the RAMs get enough refresh is slightly different in each case, although both use the GDC's internal refresh counter. The ultimate concern in both cases is to use these refresh cycles during HSYNC to refresh every RAM. The complication arises when multiple pages of RAMs share the same output bus. To effect refresh, each RAM must be given at least the refresh row address and a $\overline{RAS}$ strobe. If no more is done, the outputs will stay in the high-impedance state (if $\overline{CAS}$ has been returned to a high). This forms a "$\overline{RAS}$ only refresh" cycle with no conflict on the output bus because all RAMs have high-impedance output levels.

On the other hand, a normal cycle in a multiple page situation would give the RAS strobe only to the addressed RAM page. Some logic must be provided to assert all the RAS signals and suppress all the $\overline{CAS}$ signals during HSYNC refresh cycles in multipage systems. Single-page systems do not need this hardware since there is no danger of having multiple RAMs drive the same lines. Neither do multiple-plane, single-page systems need this extra gating because all the planes are already $\overline{RAS}$ed and $\overline{CAS}$ed for every cycle, and there are no common outputs.

Display Cycle Timing

If the memory cycle is a display cycle, the GDC plays no further part in the display process after the ALE falling edge. The video pixel clock rates can range as high as 80 MHz in some GDC applications and the NMOS circuitry in the GDC cannot possibly keep up in the realm of ECL logic. Once the display memory address is output, the GDC will raise its ALE output at the end of the E2 clock cycle. If this is a zoomed display cycle, pairs of clock cycles will then be added to the memory cycle time to stretch the cycle time while ALE remains high. See figure 2-3. The lengthened memory access time combined with an externally slowed down video shift register clock will produce the zoom magnification effect. Two clock cycles are used for the display cycle for each zoom factor increment. In other words, two clock cycles are used when zoom factor = 1 (normal video), four cycles are used when zoom = 2, etc.

Figure 2-3. Zoom Magnified Display Cycles

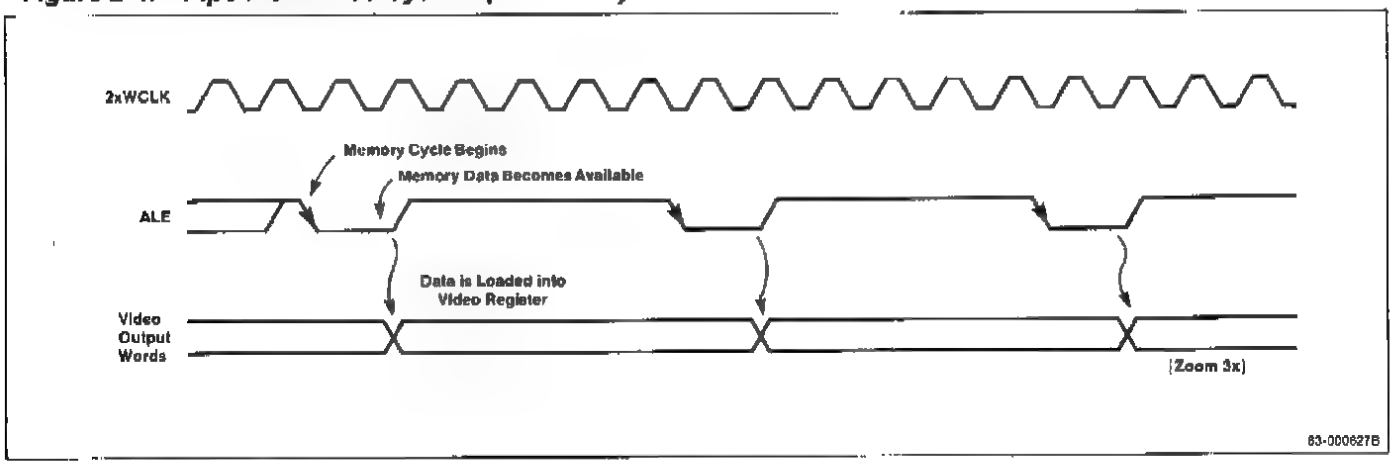


The external video circuitry must be coordinated with the GDC's memory cycle timing. To do this, visualize the overall GDC/video output system as a pipeline with several stages. This is especially important when zoom operation is planned. The video circuitry is always displaying the most recently accessed video data, while the GDC prepares the next word of video pixel data.

The moment in time when the first video register is loaded with the data just fetched from display memory is the time when responsibility for processing passes from the GDC to the video output circuitry for that data. This moment of transference is the interface between the relatively slow domain of NMOS circuitry, and the relatively high-speed world of TTL or ECL. This is also the moment when all the accumulated delays in the GDC/display memory system must be reconciled against the tight tolerance bipolar timing. Finally, this is the moment in time around which the entire video output circuitry needs more data while the last few pixels are going out to the CRT unit. Figure 2-4 illustrates this principle.

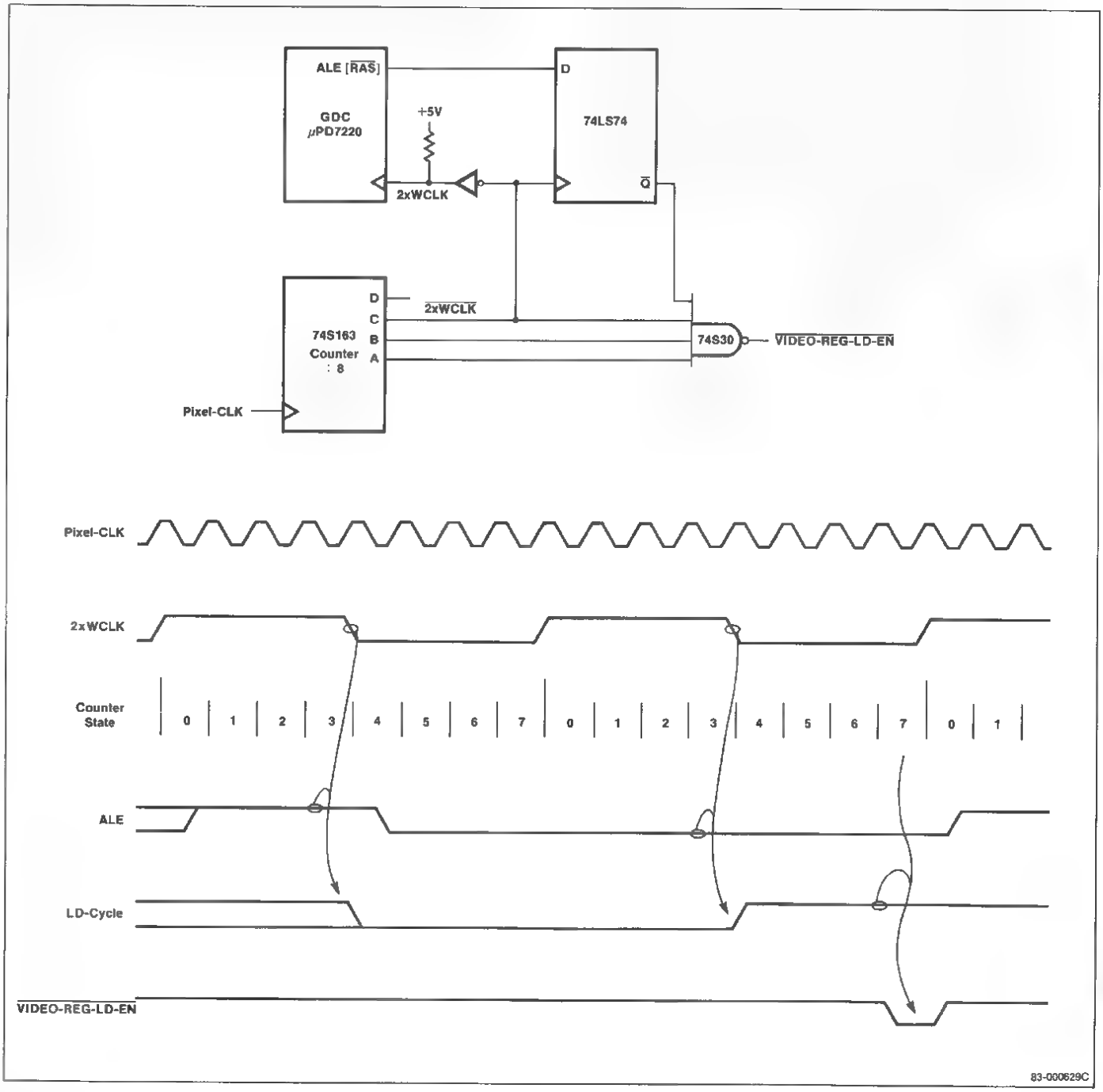
External hardware must be able to find the end of the second cycle so that the video shift register can be loaded. This is done by using the ALE signal, which will be low at the falling edge of 2xWCLK during any clock cycle in which the video shift register may be loaded. Some clock cycles in which this is true are not display cycles, but the load will be a "don't care" action, since the video output from these cycles will be blanked by the GDC anyway. "Don't care" loads can be found during RMW cycles or dynamic memory refresh cycles.

Figure 2-4. Pipelined Video System (Zoom = 3)



The hardware to time the video shift register load can be very simply implemented. See figure 2-5. The timing system starts with a clock at the video pixel rate. This is divided by eight in a counter to generate the 2xWCLK for the GDC. This counter cannot be used to differentiate between the two clock cycles that make up a display cycle without reference to the ALE signal. The terminal count of the counter, count 7, during the second 2xWCLK cycle should be used to enable the loading of the video register. By clocking the value of ALE into a flip-flop with every falling edge of 2xWCLK, the flip-flop's Q output will be high at count 7 time only during clock cycles in which the video register may be loaded. If this Q output is ANDed in a four-input gate with the three counter outputs, a load enable signal will be generated to enable the loading of the video register synchronously. At slower video rates, the carry-out of the counter can be used with a two-input gate and it will still meet the video register's input set-up time requirements.

Figure 2-5. Video Register Load Enable Circuit (High-Speed)



There is one additional hardware constraint to be considered in the design of zoom circuitry. As can be seen in figure 2-6, an abnormal ALE signal is generated if BLANK goes high while ALE is high. This problem occurs only when the zoom register is set to a value other than zero, and is characterized as the trailing edge of ALE going low one-half of a 2xWCLK cycle later than normal. As can be seen, $t_{RSL}(1)$ is 0.5 t_{CY} longer than $t_{RSL}(2)$. In the first case, ALE going low is triggered by the trailing edge of the 2xWCLK. In the second case, it is triggered by the leading edge of the 2xWCLK. This results in the 7220 specification: $t_{RSL} = 1/2\ t_{CLK} + 30$ ns minimum.

DISPLAY ZOOM HARDWARE

Introduction

The μPD7220 GDC can magnify all the information displayed on the CRT screen. It does this zoom magnification by pixel replication, in which each displayed pixel in the display memory is repeated in both the x and y dimensions the number of times which correspond to the zoom factor. Therefore, for 2x zoom magnification each displayed pixel occupies the area of four unzoomed pixels. It is twice as wide and twice as high as an unzoomed pixel. The GDC can generate zoomed displays of any interger zoom factor between 1 and 16. This capability is especially useful when using large display formats to allow easy viewing of pixel-sized details. The zoom option is available only in the bit mode graphic mode of operation.

The GDC supports all the requirements for zoomed display operation except the stretching of individual pixels. This takes place at video speeds which can be far above the GDC's frequency limitations.

Zoomed Display Cycle Timing and Implementation

An unzoomed (1x) display cycle uses two cycles of the 2xWCLK clock signal, D1 and D2. To start the display cycle, the falling edge of the ALE output pin occurs in the middle of the first clock cycle. ALE stays low until the end of the second clock cycle, when it goes high in preparation for the next display cycle. The address is output on the AD_0 to AD_{15} lines during the entire first clock cycle. The AD lines float during the second clock cycle. The video shift registers are loaded at the end of the second clock cycle (D2) with the video data just fetched from display memory. Figure 2-7 shows these relationships.

Figure 2-7. Unzoomed 1x Display Cycle Timing

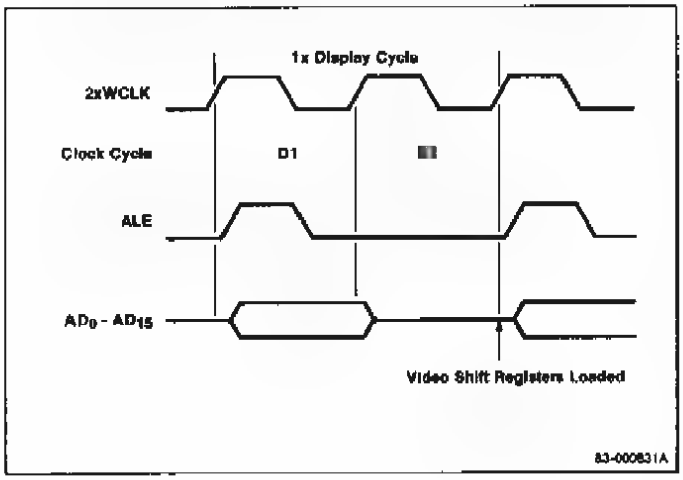
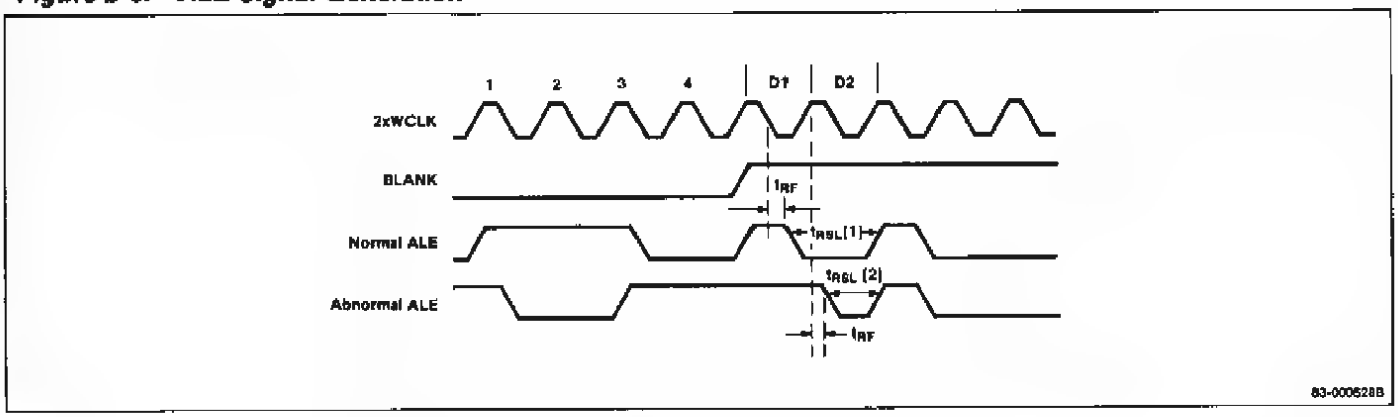


Figure 2-6. ALE Signal Generation



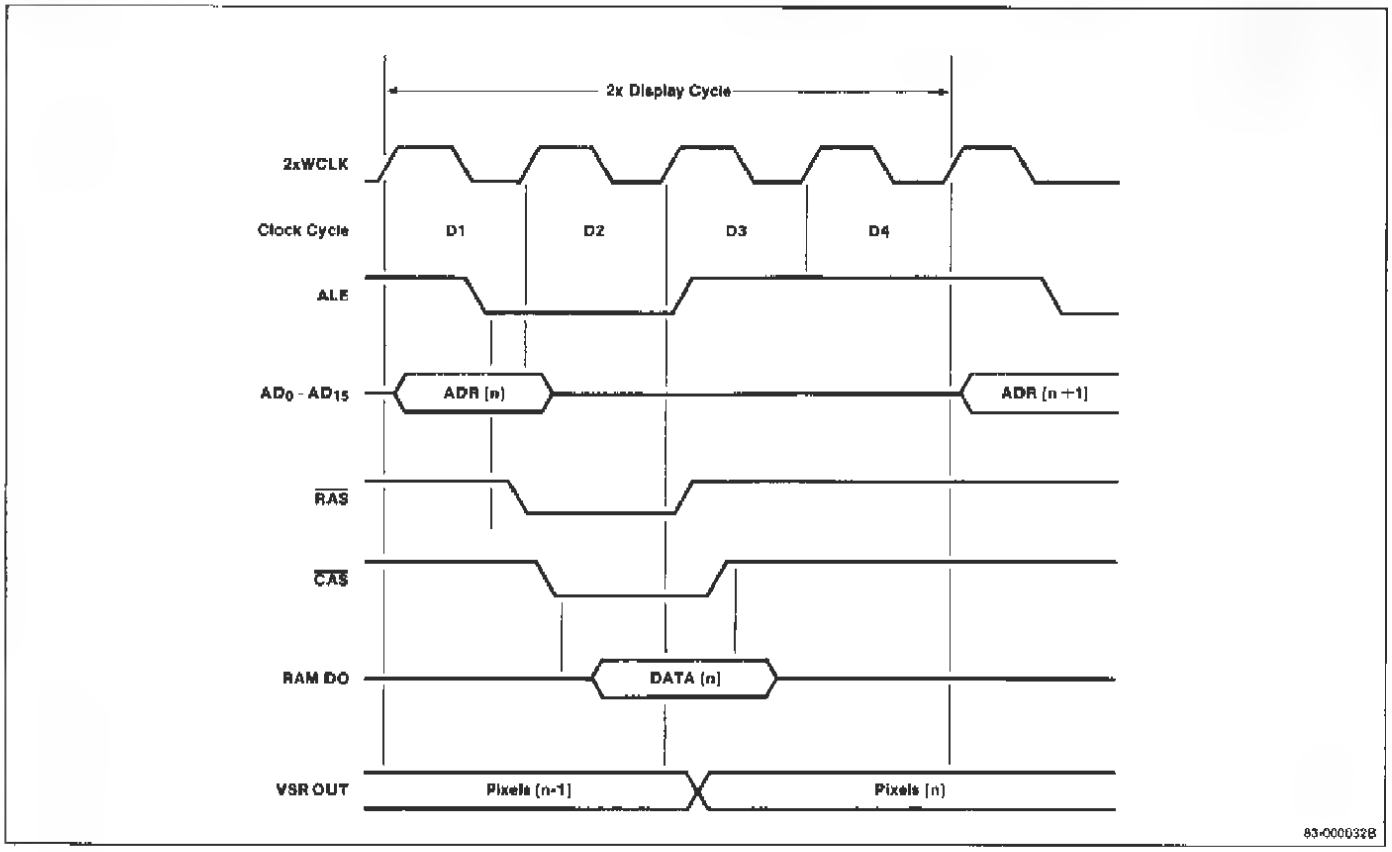
The GDC adds two extra clock cycles to the display cycle for each integer increase in the zoom factor. Therefore, 2x zoom magnification would use four 2xWCLK clock cycles, while a 5x zoom factor would specify 10 clock cycles. The timing during the first two clock cycles is the same regardless of the zoom factor. For zoom factors of two and above, the GDC marks time during clock cycles D3 and beyond with its ALE output high to indicate that the video shift registers should not be loaded during this period.

During the zoom-lengthened display cycles, the video hardware must slow down the video pixel output rate to match the zoom factor used by the GDC. In this way, each individual pixel is stretched by an integer number of pixel periods as it is serialized for the CRT display. The GDC automatically takes care of repeating the horizontal display lines to effect the vertical magnification.

An easy way to accomplish pixel stretching is to use synchronous video shift registers and control their shift enable and load inputs. This guarantees that the first pixel of each word will be of the same duration as every other pixel of the word. A synchronous counter can be used as a zoom factor prescaler to enable the video shift registers to shift only when its carry output (CO) is high. The zoom prescaler counter is set to count the zoom factor by modules so that, for example, its CO output is high for one pixel time every five pixel clock periods for 5x zoom operation. When the prescaler is loaded with all ones, its CO output will always be high and an unzoomed, 1x display will result. The counter load value must be stored by the host microprocessor in a hardware register and must be available constantly for the prescaler.

The RAM timing generator circuit suggested elsewhere in this manual generates the $\overline{RAS}$ and $\overline{CAS}$ waveforms shown in figure 2-8. Higher zoom factors than 2x add more pairs of clock cycles after D4, but there are no transitions during these cycles.

Figure 2-8. 2x Zoomed Display Cycle Timing



Note that the memory read cycle takes place while ALE is low. Triggered by the falling edge of ALE, the $\overline{RAS}$ and $\overline{CAS}$ falling edges start the RAM cycle. The data outputs of the RAMs assume the values of the stored data in time for the end of the D2 clock cycle. The data is then loaded into the video shift registers (VSR). Once ALE has gone high, the RAM precharge time is started by $\overline{RAS}$ going high, and after a sufficient hold time the $\overline{CAS}$ line is brought high to float the RAM DO lines and prepare for the next cycle.

Note that the video shift register (VSR) is loaded at the end of the D2 clock cycle, as it was at 1x zoom. Although the GDC starts its cycle with the D1 cycle, the video output hardware begins outputting a new word of video data at the end of D2, regardless of zoom factor. An effective way to think of this relationship is to imagine the GDC anticipating the need for new video data for the video shift register two clock cycles early. It then proceeds to do the proper memory cycle and idles until two clock cycles before the next shift register load.

The moment the video shift registers are loaded is critical in the timing scheme. Because it is the start of the video output hardware cycle, everything must coordinate at this point. Not only must the shift registers be loaded at just the right moment, but the pixel stretching prescaler counter must be initialized to its load point so that the first pixel will be stretched the proper amount. This can be done by ORing the CO of the zoom prescaler counter with the video shift register load enable pulse and feeding the low true result into the synchronous load input of the zoom prescaler (a 7402 into a 74163, for example).

Using the approach outlined above, zoomed displays can be implemented using little extra logic. One 16-pin DIP houses the zoom prescaler counter (74x163), the load gate requires 1/4 14-pin DIP (7402), and the storage register for the hardware zoom factor can use 1/2 of an octal latch (74LS273). The zoom logic is diagrammed in figure 2-9.

WIDE DISPLAY ACCESS MODE

Introduction

The graphics display controller can be programmed to work with a double word access in display memory during active display cycles. In graphics mode, 32 bits of video data are accessed simultaneously, instead of the normal 16 bits. Video data rates up to 80 MHz are then possible using a 5-MHz GDC clock. Note that during read-modify-write cycles, a single 16-bit word is processed by the GDC, regardless of the display cycle mode.

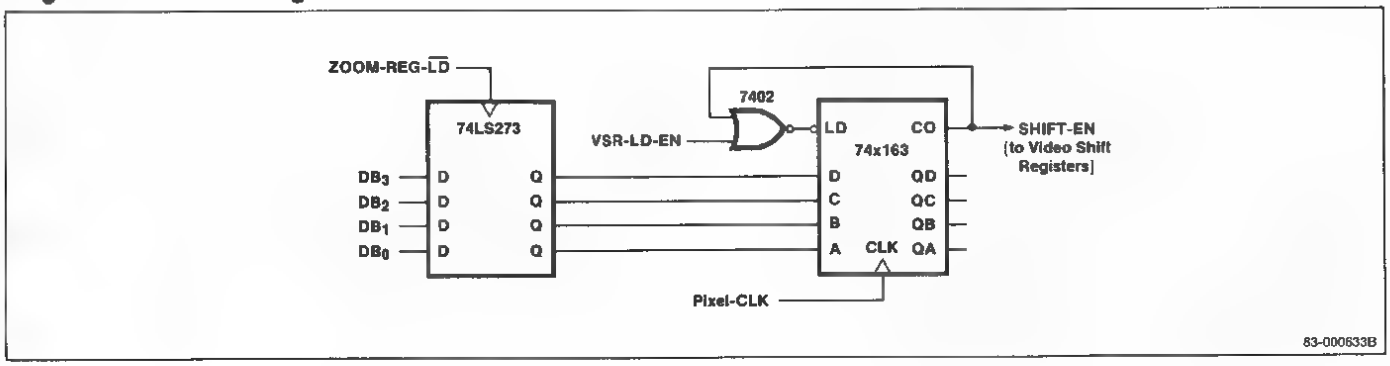
The wide display accesses can also be used in character mode and mixed mode. No matter which mode is used, the width of the display access is always two display memory words. Note that with character mode's 13-bit word width, the wide access mode would provide a 26-bit double word display access. To simplify the discussion, reference will be made only to the 16/32-bit accesses of graphics and mixed modes.

The GDC supports double-word display cycles by generating the address of every second location to be displayed. To find the next word address, the GDC's display word address counter is incremented by two instead of the normal mode's one. External hardware must then access and video serialize both the even and odd words at these successive addresses. It is generally necessary to have separate banks of RAMs for the even and odd words in display memory so that all 32 output bits can be accessed and loaded into the video shift registers simultaneously.

The wide display access mode is controlled by a bit in the parameter RAM for each display area. If bit 7 of PRAM bytes 3 or 7 (11 or 15 in character mode) is set, the display address counter will be incremented by two for each display cycle in the corresponding area on the screen.

Wide display access mode is useful when generating ultra-high-speed video, and for coded characters, when using up to 32 bits per character (26 bits in character mode).

Figure 2-9. Zoom Logic



Structure of Display Memory

In the straightforward implementation of wide display access mode hardware, each plane of display memory must have two separate banks of RAMs. The data output pins of each bank must be kept separate to allow the desired 32-bit accesses in each plane. Each bank of RAMs, two for each plane, must have its own RMW cycle read buffer and 16 bits of video shift register. The video shift registers of the even and odd banks of each plane must be tied together to produce a 32-bit shift register.

The decoding of the address from the GDC must use A_0 to differentiate between the even and odd banks of RAM during RMW cycles. The highest significant address bits then differentiate the planes. During RMW cycles, the bank of RAMs to be enabled onto the bus during DBIN time and then written into is selected using A_0 and the plane select bits. During display cycles, the least significant bit and the plane select bits are ignored while video data is sent to the CRT screen.

The strobes for each bank of RAMs can be handled similarly to the normal 16-bit access mode. If no banks of RAMS have multiple groups of RAMs (no outputs tied together), all banks can be RASed and CASed each memory cycle, blanked, active, or RMW. If there are wire-ORed RAMs, it may be desirable to do RAS-only refresh cycles during HSYNC time so that all RAMs get all the dynamic RAM refresh cycles.

In straightforward systems, the minimum size of a plane of display memory in wide display access mode is twice the size of a bank of RAMs. If 64K-by-1 RAMs are used, the minimum plane size would be 128K 16-bit words. It is possible to use very fast RAMs and make two display accesses during the two 2xWCLK cycles of a GDC display cycle (as short as 400 ns). In such a system, both odd and even planes are located in the same 16-bit memory chip set. This would have the advantage of a smaller minimum plane size, but would require more complex control circuitry.

Ultra-High-Speed Video Hardware

One of the reasons for using the wide display access mode is to obtain ultrahigh video data rates. With a 5-MHz 2xWCLK, the video data rate will be 80 MHz, or 12.5 ns per pixel. At these speeds, ECL devices should be used for the video shift registers. The conversion from TTL to ECL levels is easily done at the outputs of the RAMs before loading into ECL shift registers. This technique, although conceptually simple, requires 32 level translators per plane and a large number of ECL shift registers.

A different approach is to use low-speed TTL shift registers at the outputs of the RAMs, which are clocked at a fraction of the output video rate. Their outputs are converted to ECL levels and loaded into one high-speed ECL shift register per plane. The ECL shift register is clocked at the full video data rate. If four 8-bit TTL shift registers are used with a high-speed ECL 4-bit shift register, fewer level converters will be necessary. Of course, the timing circuitry will be somewhat more complicated to support this two-stage shift register configuration. The oscillator and high-speed timing must be done at the pixel clock rate and will be likely to use ECL circuitry.

32-Bit Characters

When very large sets of characters must be displayed with numerous attributes using a coded character, 16 bits (or 13 bits) per character are often insufficient. The wide access mode can be used in these situations to provide up to 32 bits per character. Conceptually, the two 16-bit halves of each character are accessed by separate RMW cycles. The 32 bits accessed during display cycles are used in parallel by the character generator and attribute logic, along with the character row line count, to scan out the character. This capability is especially important when displaying oriental character sets or multiple fonts of characters.

IMAGE BIT

Introduction

An image bit is stored in the GDC's parameter RAM for each display area partition (bit 6 in PRAM bytes 3, 7). The value of the image bit is set with the PRAM command which loads the appropriate parameter RAM bytes. In general, the value of the image bit is set at initialization and remains unchanged as long as the display area type does not change. These bits are important in the mixed mode of operation but have an effect in other modes also.

In mixed mode, a display area's image bit specifies whether the area is a bit-mapped graphics area or a coded character area. In differentiating between these area types, several changes occur in the way the GDC operates. First, in a coded character area, the lines per row (LR) parameter of the CCHAR command controls the number of times a given line of words is scanned while the line counter is incremented. In a bit-map area, each line is automatically scanned only once. Second, in a coded character area, the external circuitry must include a line counter and a character generator. In a bit-map area, the external circuitry must use only a video shift register. The GDC outputs the value of the image bit during HSYNC time on the A_{17} pin, where it should be loaded into a flip-flop to control the external circuitry.

Third, the way the GDC handles the pattern register during RMW drawing cycles differs between the two types of areas. In a bit-map area, the pattern is used one bit at a time, while in a coded character area, all the bits of the pattern are used in each RMW cycle.

The fourth difference is the nature of the information flow to the screen in each type of area. In a bit-mapped graphics area, 16 pixels are sent to the CRT from each word access. In coded character areas, each word access gets a character code value which is converted to a character window's width of pixels by an external character generator. To mix characters and graphics on the same screen, it is desirable to use the same pixel clock rate for both areas and to have a consistent number of pixels on every line, regardless of the type of information being displayed. To make this mixing easier, the GDC assumes that the character windows will be eight pixels wide. (There are no physical limitations here, just compatibility considerations. The GDC can be used in mixed mode with some other window width if no graphics will be displayed.) Therefore, two character windows equal one graphics 16-pixel access width.

The GDC needs two 2xWCLK cycles to access a word in display memory. In mixed mode, one character code and its attributes are accessed during this minimum cycle time. To make 16-pixel graphics accesses compatible with 8-pixel character accesses, the GDC accesses a new graphics word every four 2xWCLK cycles instead of every two. Therefore, each pair of 2xWCLK cycles equals eight pixel times, regardless of the type of display area.

During initialization, the AW value must be set for the number of 2xWCLK cycle pairs to be displayed, not the number of graphics words.

Timing Details

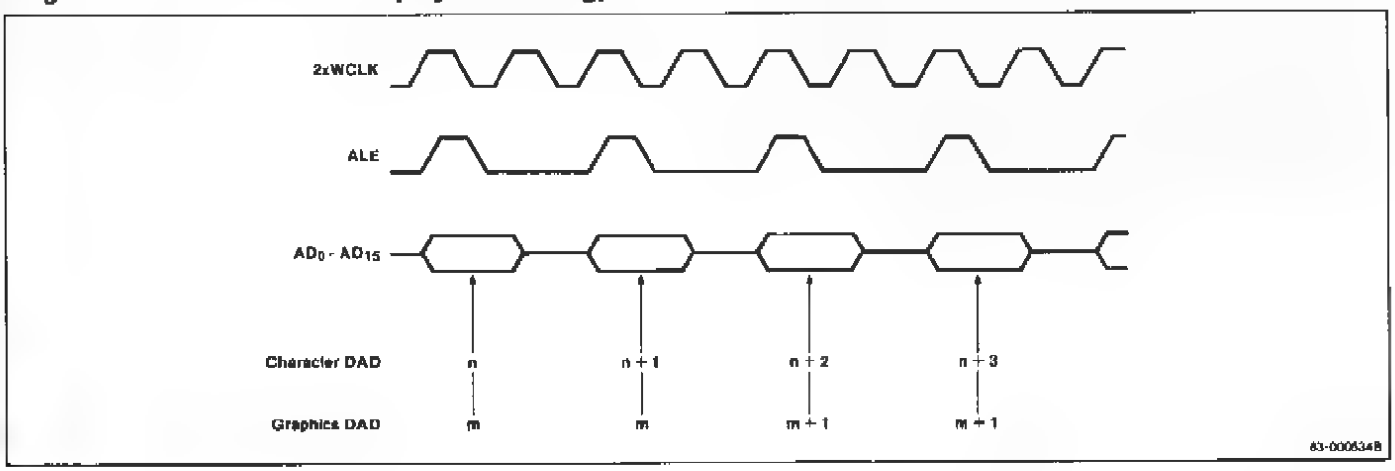
Two timing situations can occur during mixed mode operation. Both use the same basic two 2xWCLK cycle interval. In the first, a coded character area uses just one of these intervals to make one coded character access. The display word address, DAD, is then incremented for the next character address. In the second, a bit-mapped graphics area uses two of these intervals, or a total of four 2xWCLK cycles, between increments of the display word address. In either case the GDC generates an ALE cycle every two 2xWCLK cycles. Figure 2-10 illustrates this principle.

A design using both coded characters and bit-mapped graphics will need to detect when an image (graphics) area is being displayed and suppress the second, fourth, etc loads of the graphics video shift registers. This allows all 16 bits of the first access at each address to be properly serialized and sent to the CRT.

The GDC starts off each line in a graphics area with the first of a pair of accesses to the first word address to be displayed.

An incidental advantage of mixed mode coded character and bit-mapped character compatibility is that the 2xWCLK runs twice as fast in a mixed mode graphics area as it would in graphics mode to generate the same display. This fact is very useful at the slower display speeds where the graphics mode 2xWCLK frequency might otherwise be lower than the GDC's minimum clock frequency. In addition, RMW cycles still take four 2xWCLK cycles, which in this mode is equal to one display fetch of 16 pixels, instead of the graphics mode's two display refresh fetches. Therefore the drawing speed is twice as high in mixed mode as it is in graphics mode for the same video display format.

Figure 2-10. Mixed Mode Display Addressing, DAD



The Image Bit in Graphics Mode

If the image bit and the GD bit are set to one in graphics mode, the GDC increments the display address after every other display cycle instead of every display cycle. As this bit is not output over the A_{17} pin, all areas of the screen should be consistent in the use of this capability. There is no way to detect where one display area ends and another begins in graphics mode. If normal graphics mode operation is desired, the image bit and the GD bit for each display area must be set to zero to force an increment of the DAD for each pair of 2xWCLK cycles (one display cycle).

SMOOTH HORIZONTAL PANNING

The GDC can start raster-scanning the display memory at any word address. The GDC has time to calculate and output the sequence of addresses necessary to generate the raster-scan video signals. The GDC is not involved with the actual conversion of the data accessed from memory into the appropriate video signals. The frequencies involved in this process can be as high as 80 MHz, which is far above the capabilities of NMOS technology. It is often desirable to be able to position the display window at any bit position and not be constrained to word boundaries. External hardware can easily do this at video pixel speeds. When using smooth panning, zoomed display capability is lost.

A straightforward approach to smooth panning within word boundaries uses a serial-to-parallel shift register downstream of the normal parallel-to-serial video shift register. The extra shift register must be a full display cycle width of pixels long. This register is clocked at the video pixel rate and acts like a digital delay line for the serialized video data.

To support smooth panning, the GDC must be programmed for an AW count two words greater (AW must be an even number) than the desired display width so that, during smooth panning, part of both the first and last words can appear on the screen simultaneously within the desired display window. The blanking signal from the GDC must then be delayed by two display cycle periods at the start of the active display area. Blanking must not be delayed at the end of the line.

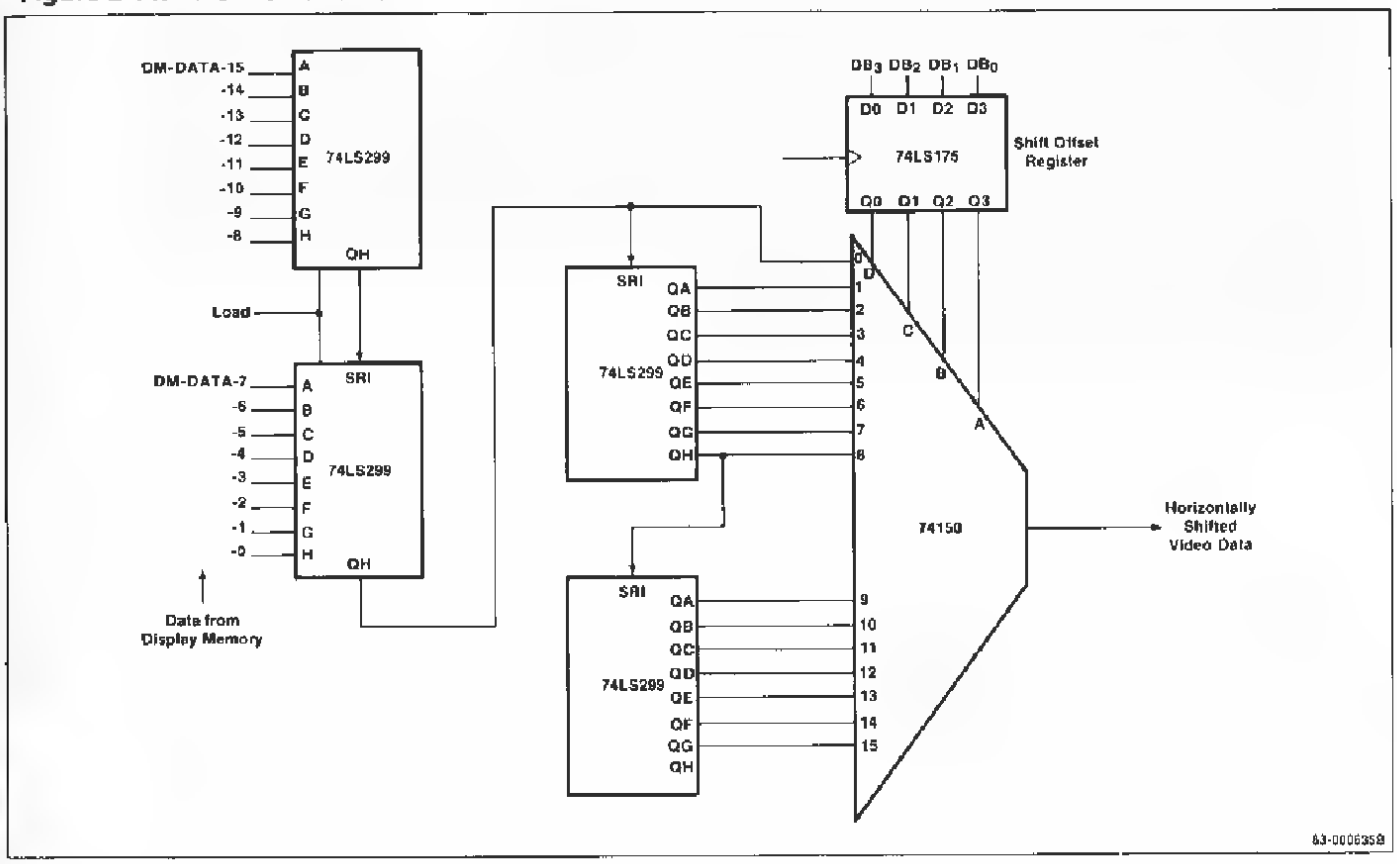
The proper delay-line shift register tap must be selected by a multiplexer under the control of the host processor. The tap is advanced pixel by pixel to smoothly pan the display. The first multiplexer input must be connected to the regular video shift register's output. Each following input is connected to the next "more delayed" delay line shift register output.

The first two active display cycles of each line fill up both shift registers with video data. The first word accessed is lost. The second word accessed occupies all the bits of the delay-line shift register at the moment blanking is removed from the video stream. The particular bit selected by the multiplexer will be the first bit seen on the left edge of the screen. This word will appear to be bit-shifted to the left, behind the blanking edge. This apparent shift will continue across the screen for all the other accesses on the line.

The extra logic necessary to do smooth horizontal panning includes the delay-line shift register, the multiplexer, and a storage register to hold the delay offset select bits for the multiplexer. (Refer to figure 2-11.)

It should be noted that it is not possible to support both smooth horizontal panning and zooming options at the same time. This is due to the effect of pixel stretching, which also occurs during the horizontal front and back porches. Take the case where zoom is set to 16 and the smooth horizontal panning circuitry is in place. If the horizontal back porch is set to five words, the horizontal front porch is set to five words, and the number of active words per line is set to 40 (640 pixels per line), both the front and back porches will occupy a space of 1,280 pixels rather than their normal width of 80 pixels. It is quite clear that the additional 2,400 blanked pixels per line will result in a completely blanked raster line.

Figure 2-11. Horizontal Smooth Scroll Hardware



READ-MODIFY-WRITE CYCLES

Read-modify-write (RMW) cycles are used exclusively by the GDC when it accesses its display memory for data transfers. Various activities like clearing memory, figure drawing, DMA transfers, and reading and writing memory contents all use RMW cycles. The timing used is different from display cycle timing. Instead of just supplying an address, the data accessed by the $\overline{RAS}$/$\overline{CAS}$ sequence must be read into the GDC, modified by the GDC's logic unit, and the result written back into the memory.

These additional activities are coordinated with the GDC's $\overline{DBIN}$ output and the 2xWCLK. The basic RMW cycle takes four clock cycles, E1 through E4. When the display zoom magnification factor is larger than 2, the RMW cycle is extended with ALE high to be the same length as the zoom lengthened display cycle. It is important to note that an RMW cycle always includes the write back operation, even when the data is not changed but just read. This technique requires only one operating mode for the external hardware, to help keep it simple. The GDC will echo the display memory location's contents as it was read when there are no changes, so that the effect of the write operation will be to leave the display memory location's contents unmodified.

An RMW cycle (figure 2-12) starts off just like a display cycle, with a falling edge of ALE in the middle of the E1 cycle. The display memory address must be output to the display memory as it always is. In fact, at this point there is no way to tell an RMW cycle from a display cycle. The first difference occurs when $\overline{DBIN}$ goes low just after the middle of the E2 cycle. The GDC is thereby requesting the read data from the display memory to be placed on the AD lines so that it may be input by the GDC at the middle of E3. The $\overline{DBIN}$ low period lasts until just after the middle of E3, at which time it returns high.

The data is modified during the second half of E3, and the resulting data becomes available on the AD lines with the start of the E4 cycle for write back, and is stable through the end of E4. Note that ALE stays low until the end of E4, and therefore the video shift registers will be loaded at the end of E2 (as in a display cycle), E3, E4. Although these extra loads put garbage data in the video registers, no harm is done because the video will be blanked during the duration of the RMW cycle.

Figure 2-12 RMW Cycle Timing

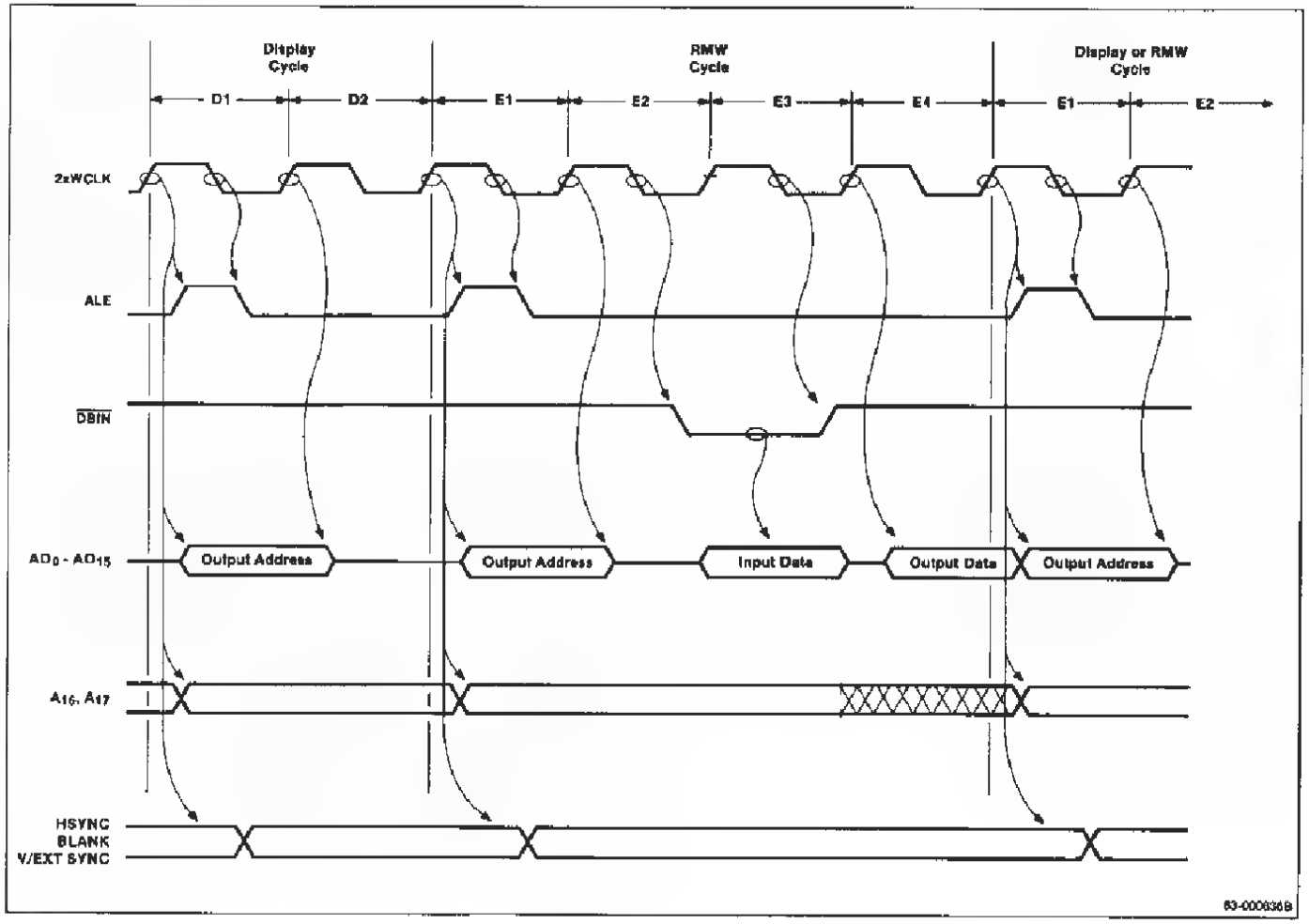


Figure 2-13 shows an example of some logic which will implement a simple system's RMW cycle timing. All the extra timing required for RMW cycles is derived from $\overline{DBIN}$ and the 2xWCLK. In simple systems the $\overline{DBIN}$ signal can be routed directly to the selected plane of RAMs to enable its bus buffer onto the AD bus. In addition, the $\overline{DBIN}$ pulse can be synchronized to the falling edge of 2xWCLK and delayed one 2xWCLK cycle so that it appears during E4. If this signal is then gated with $\overline{2xWCLK}$, a write enable ($\overline{WE}$) pulse can be generated that lasts for the second half of E4 only. This enable is then steered to the proper group of RAMs to enable the write operation which finishes off the RMW cycle. Systems that make more complex use of the AD bus will need more advanced timing with more carefully placed edges.

During zoomed display operation, RMW cycles are extended to the same length as display cycles if the zoom factor is three or larger. The reason for this is that RMW cycles and display cycles must interleave during real-time. Since an RMW cycle must be finished once it has begun, and a display cycle cannot be started late, RMW cycles can never be shorter than the display cycles. The GDC automatically stretches the RMW cycles out to the same length as display cycles, if necessary, by bringing ALE HIGH at the end of E4 as usual and keeping it there until the proper starting time of the next display cycle. Figures 2-14 and 2-15 illustrate this operation.

Figure 2-13. RMW Timing Generator

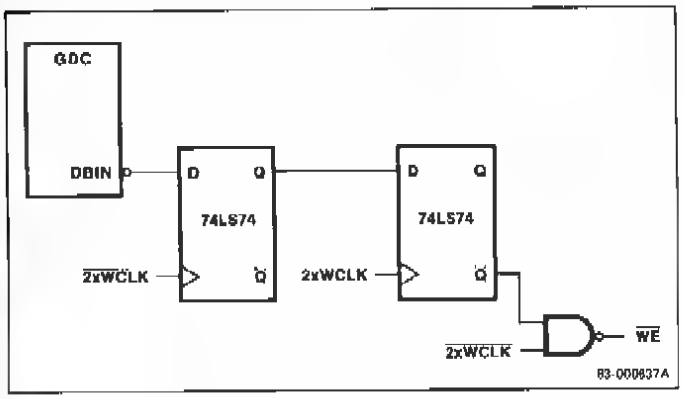


Figure 2-14. Zoomed Display RMW Cycles (Zoom = 2)

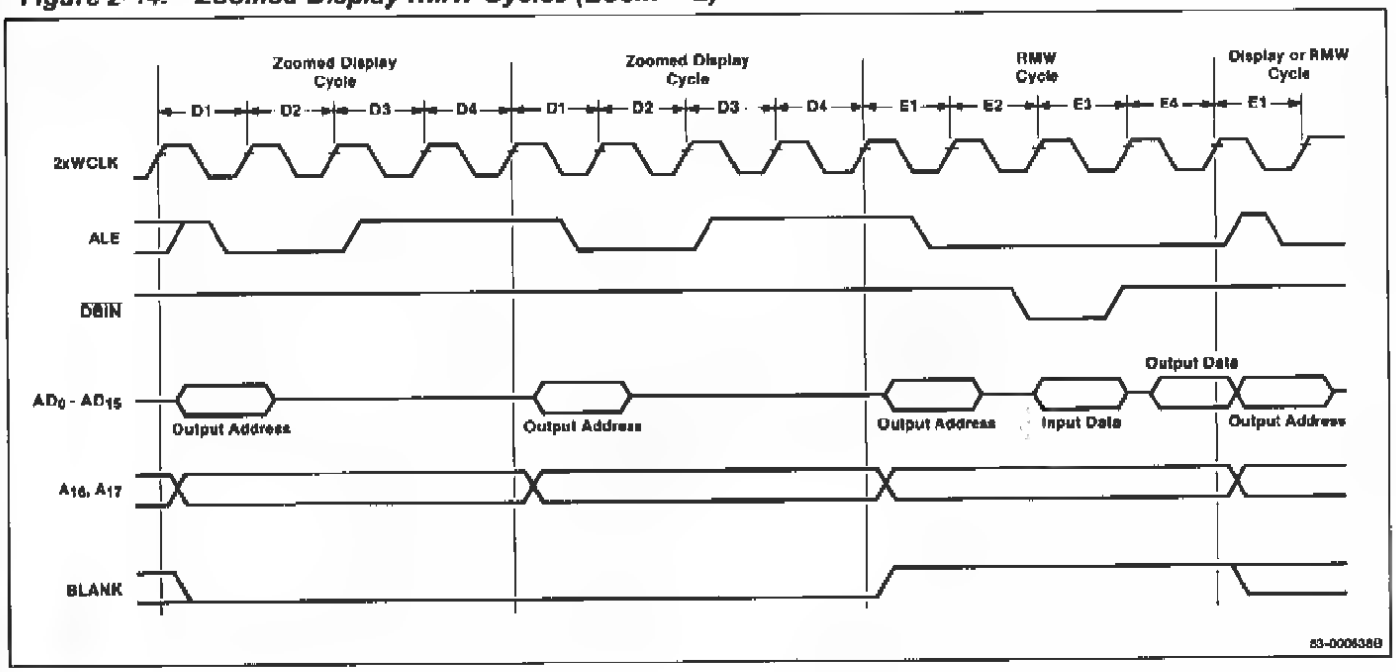
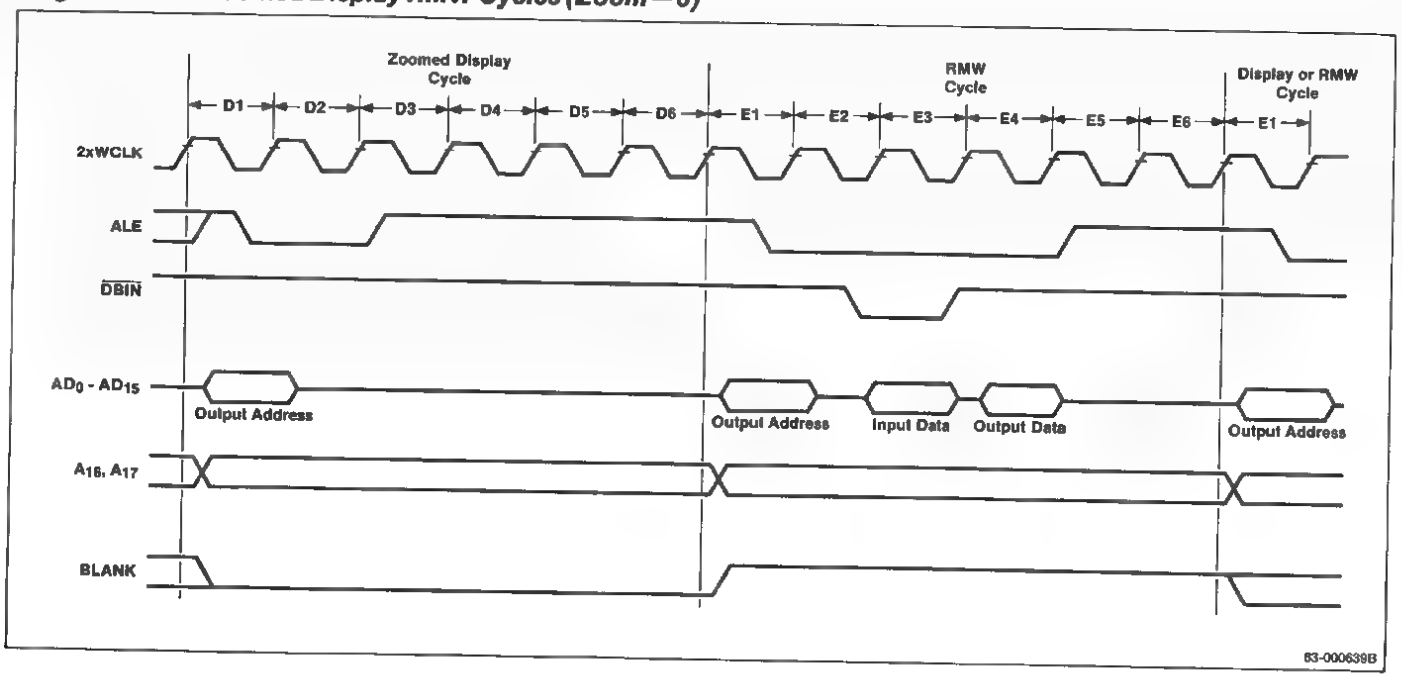


Figure 2-15. Zoomed Display RMW Cycles (Zoom = 3)



Overview

Read-Modify-Write (RMW) logic is used in every operation involving modification of data in the bit map memory (video buffer). With the GDC, you can perform a logic operation on bits being modified, you can write single or multiple bits in each RMW cycle and, in conjunction with the vector drawing processor, draw a number of patterns in the video memory. The RMW logic operates somewhat differently in the graphic and character modes. Each of the modes will be explained separately.

During a figure drawing process, the GDC must modify a number of bytes in the video memory. In each RMW cycle, only one individual bit is modified. Since the GDC can access only 16-bit words, each RMW cycle consists of:

- Read 16-bit data pointed to by EAD.
- Modify the bit pointed to by dAD.
- Write 16 bits back into video memory.

In the character mode, all 13 bits representing the character and its attributes are modified simultaneously, so that the RMW cycle consists of:

- Read 13 bits of character (16 bits in mixed mode).
- Modify some or all bits.
- Write all the bits back into video memory.

Block Diagram

The block diagram of the RMW logic is shown in figure 3-1. A few major functional blocks are directly involved in each RMW cycle.

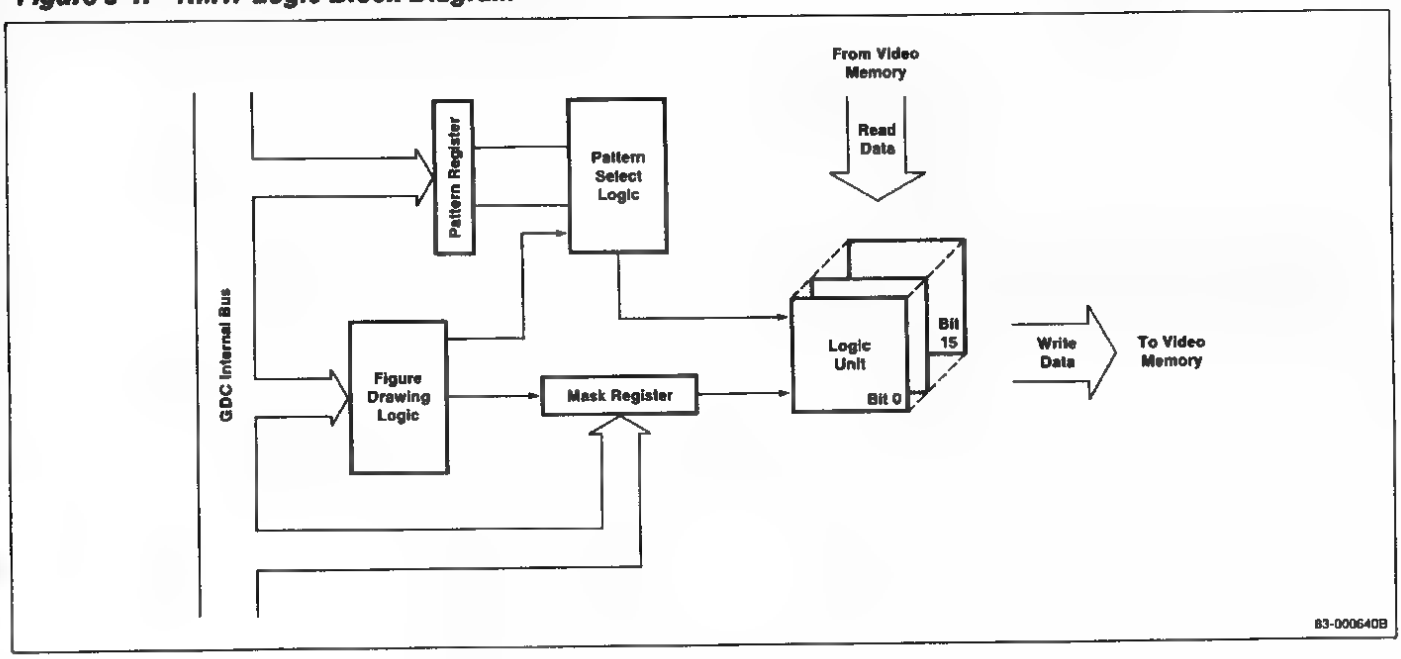
Logic Unit. This logic part does actual RMW data modification. It uses the data from both the pattern and mask registers. Depending on bit patterns, it modifies the 16-bit words from the video memory.

Pattern Register and Pattern Select Logic. These blocks select the bits on which the logic operation will be performed. During the figure drawing process, the pointer moves the LSB to MSB of the pattern register. If the bit is set to one, the logic operation takes place. If it is a zero, the bit is not modified. In the character mode, the pattern register holds either the new character or just the few bits that should be modified in the video buffer. Also, during WDAT and DMAW commands, the content of the pattern register is written into video memory in 16-bit words.

Figure Drawing Logic. This is the hardware, consisting of a number of adders, shift registers, and counters, that calculate the address of each consecutive pixel during the figure drawing process. It controls the bit selection from both the pattern and mask registers.

Mask Register. This register serves as a pointer for figure drawing logic to select which bit of the 16-bit word must be modified. As the figure drawing logic calculates the consecutive addresses of the figure it is drawing, the mask register is shifted right or left accordingly.

Figure 3-1. RMW Logic Block Diagram



LOGIC UNIT

Let us start with the logic unit hardware. The GDC offers four logic modes: REPLACE, COMPLEMENT, CLEAR, and SET.

Modes are selectable by the MOD field in WDAT and DMAW commands. The operation of each mode is explained in figure 3-2. The pattern register is loaded with a 16-bit data pattern. Before the first pixel of the figure is written into video memory, a pattern register pointer points at bit 0, and the MOD register selects one of the four types of logic operations.

In REPLACE mode, the data in the memory will be replaced with the data in the pattern register. In COMPLEMENT mode, any bit that is set to 1 in the pattern register will complement the appropriate bits in the pattern the GDC is drawing. In SET or CLEAR modes, only bits that are set to 1 in the pattern register will be set and cleared respectively during the drawing process.

As each consecutive pixel is drawn, the pattern register pointer is shifted right by one pixel. After reaching bit 15 (MSB), the pointer goes back to bit 0 and starts the pattern again. It continues this process for as many pixels as there are bits in memory to be written, regardless of the direction and shape of the figure to be drawn—vector, arc, or rectangle.

Figure 3-3 describes the RMW process. The RMW cycle starts by reading 16 bits of data from the address indicated by EAD. All 16 bits enter ■ logic unit.

The SET, CLEAR, and COMPLEMENT logic operations work similarly. Only the inputs to MUX1 differ. Figures 3-4, 3-5, and 3-6 show the data path of one of sixteen bits from video memory.

In figures 3-4 and 3-5, the particular bit is not modified. In figure 3-6, the bit is modified as it passes through MUX1. In this case, both pattern and mask register bits must be set to 1.

For the REPLACE operation, the mask register points to the bit that must be replaced. If it is zero, the data is not modified (figure 3-7). If the mask register bit is equal to one, the data that is written back into video memory comes from the pattern register (figures 3-8 and 3-9).

Figure 3-2. Pattern Register Operation During Different Logic Modes

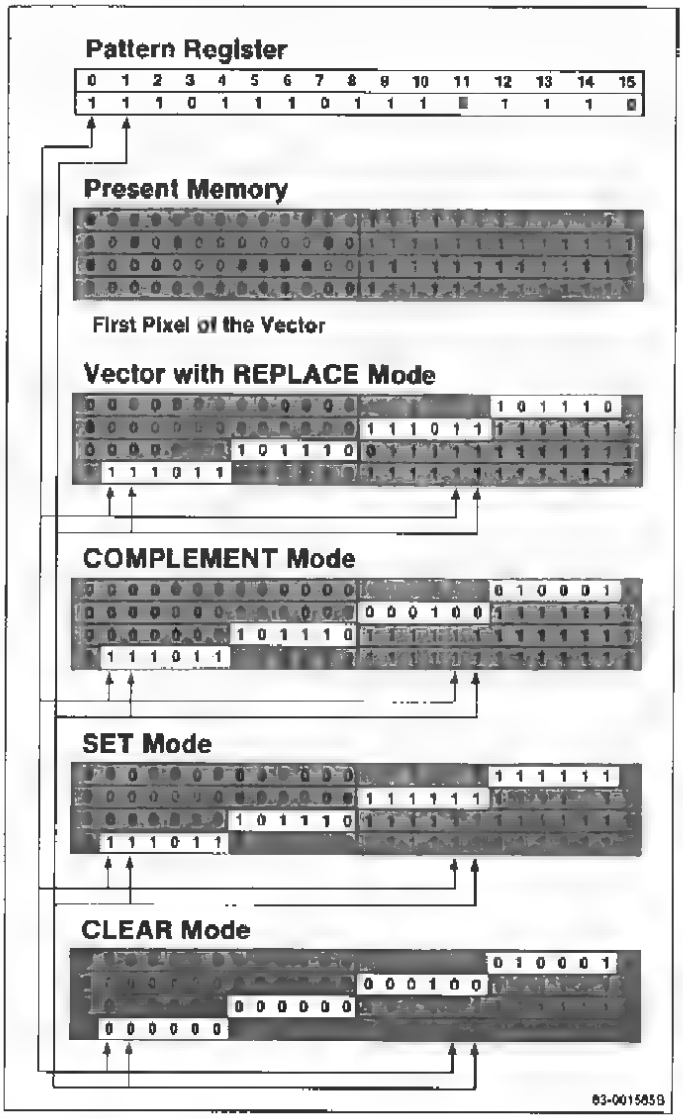


Figure 3-3. RMW Logic Block Diagram

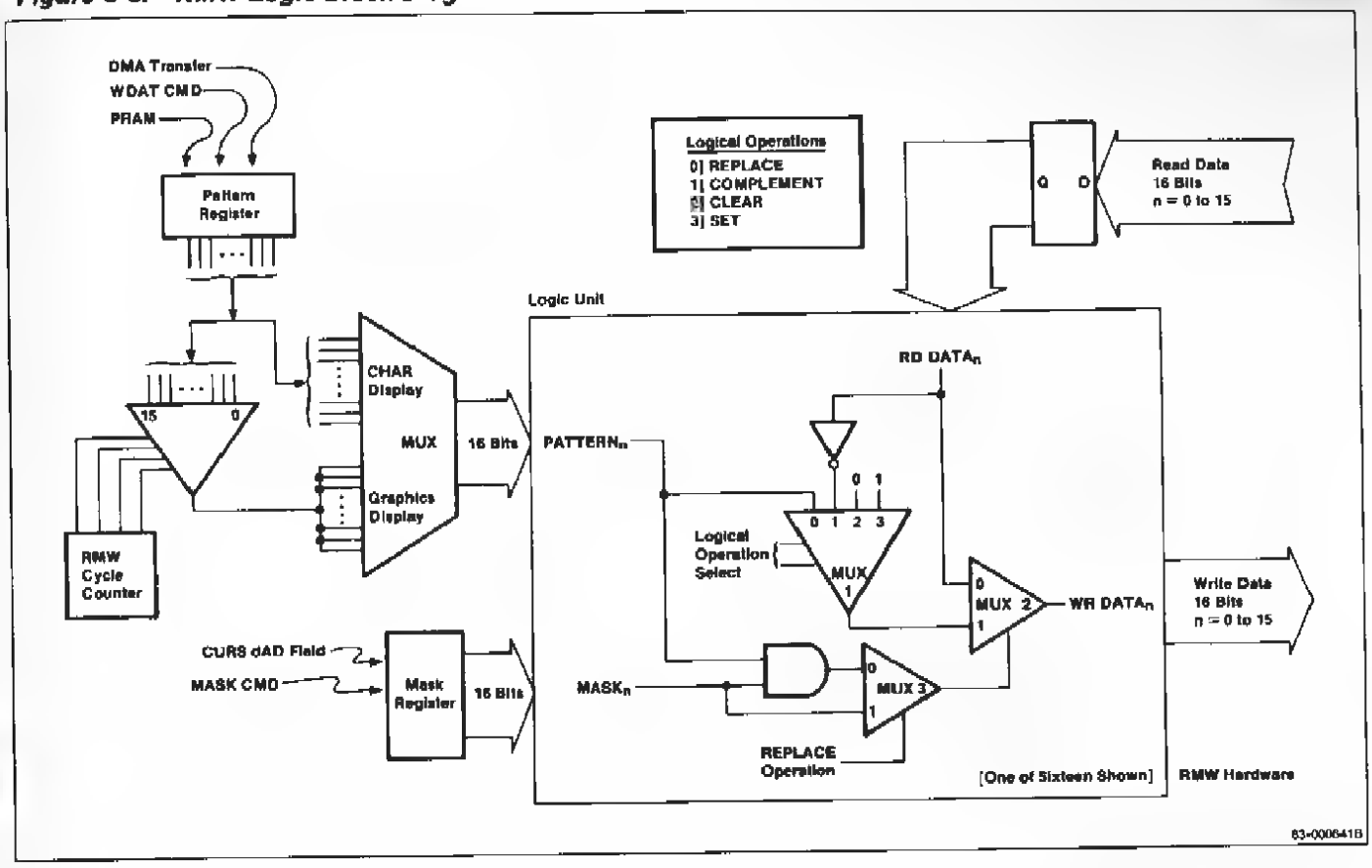


Figure 3-4. RMW Operation with: $PATTERN_n = 0$, $MASK_n = X$, Logic Operation = 1, 2, 3

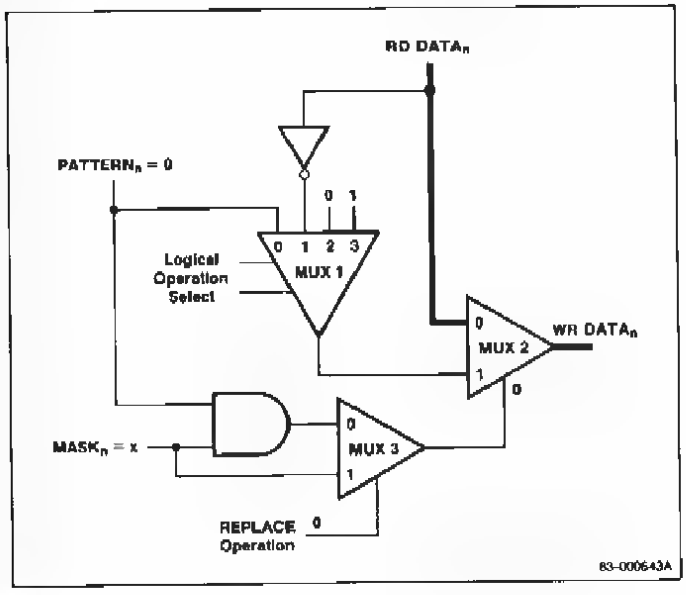


Figure 3-5. RMW Operation with: $PATTERN_n = 1$, $MASK_n = 0$, Logic Operation = 1, 2, 3

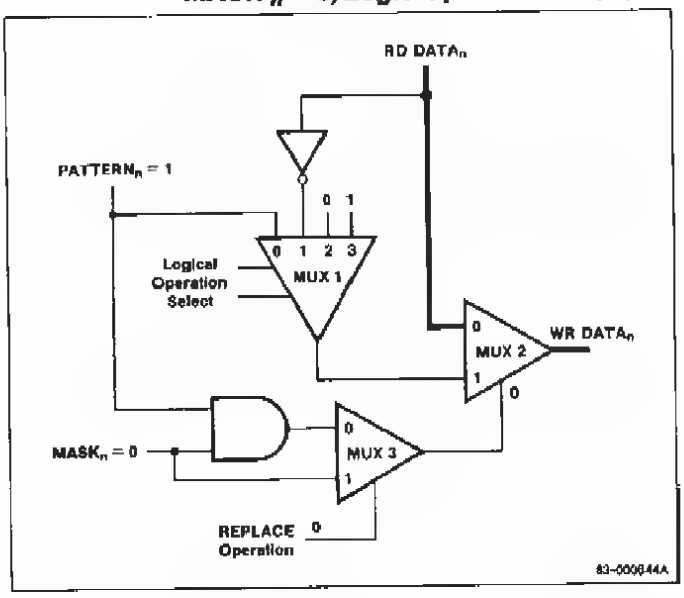


Figure 3-6. RMW Operation with: PATTERN$_n$ = 1, MASK $_n$ = 1, Logic Operation = 1, 2, 3

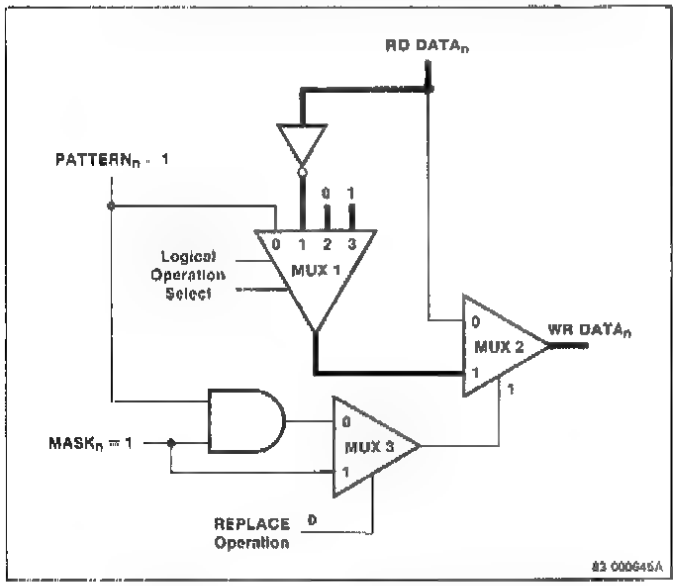


Figure 3-7. RMW Operation with: PATTERN$_n$ = X, MASK $_n$ = 0, Logic Operation = 0

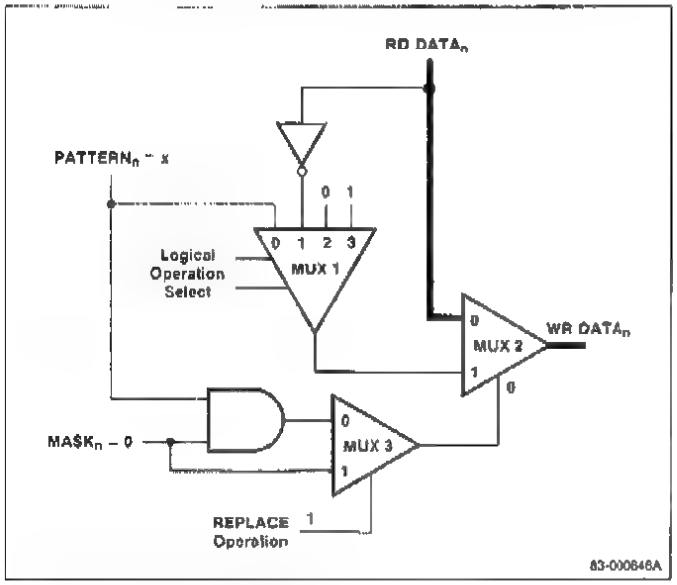


Figure 3-8. RMW Operation with: PATTERN$_n$ = 1, MASK $_n$ = 1, Logic Operation = 0

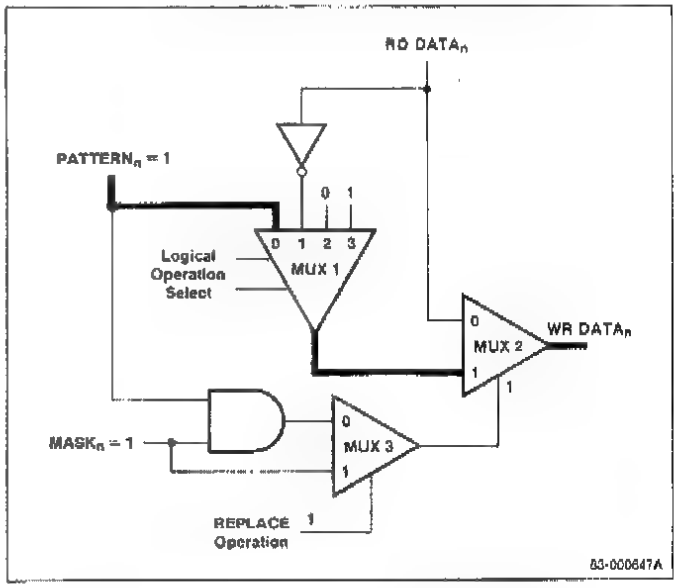


Figure 3-9. RMW Operation with: PATTERN$_n$ = 0, MASK $_n$ = 1, Logic Operation = 0

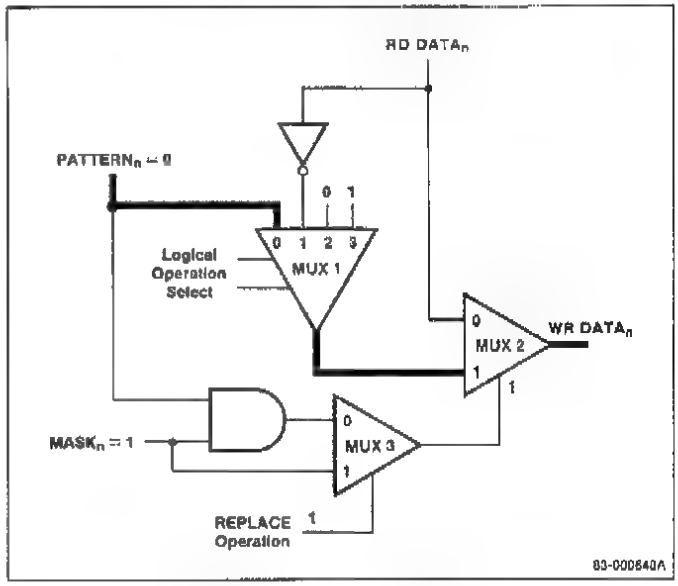
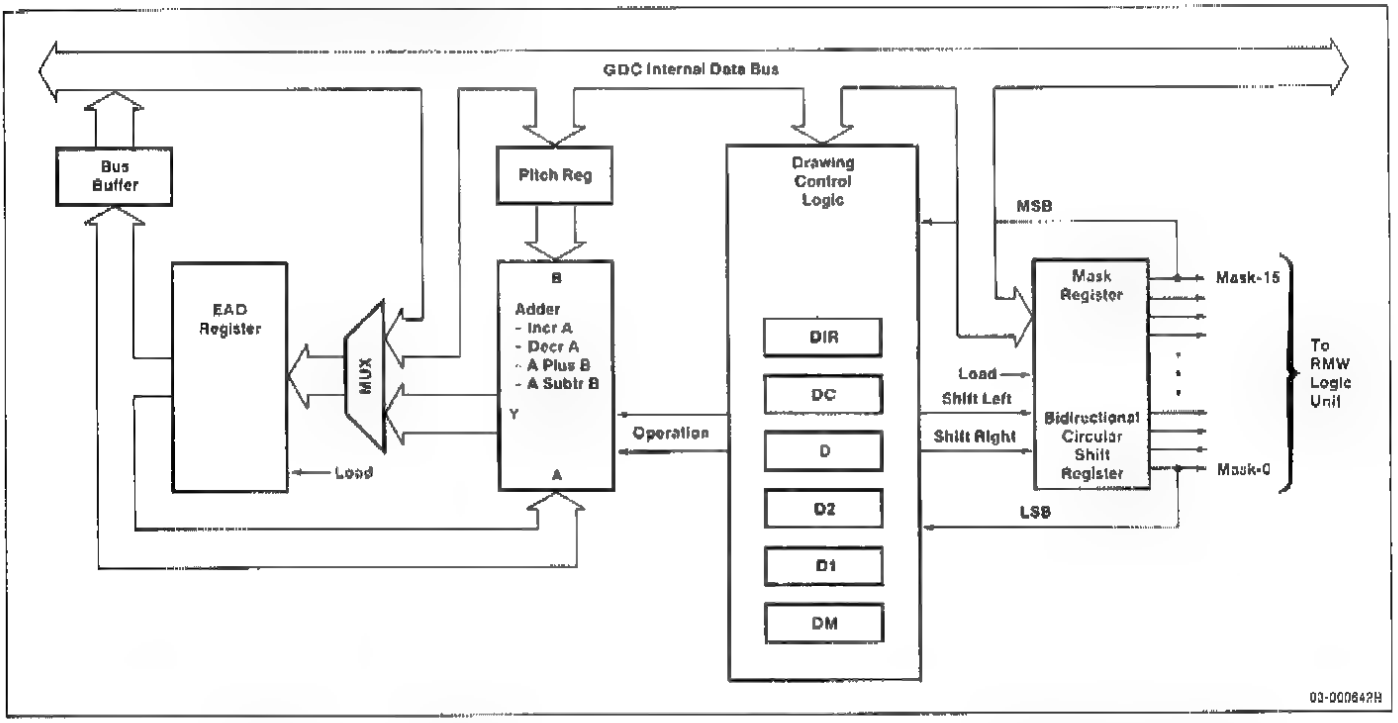


FIGURE DRAWING LOGIC (FDL) AND MASK REGISTER

This part of the logic is responsible for calculating the addresses of the RMW cycle. See figure 3-10. Before drawing is started, the EAD Register is loaded with the 18-bit address using CURS command. The mask register is loaded with the address of the first pixel of a figure. This address comes from the 4-bit dAD field of the CURS command and it is decoded into a 16-bit value with only one bit set. The drawing control logic consists of six registers that are loaded with appropriate values during FIGS command. After FIGD command is issued to the GDC, the first address is output to the video memory. The data goes into the GDC, is modified by the logic unit, and written back to the video memory.

During this RMW cycle, the FDL calculates the next address. It manipulates the values in D, D1, D2, and DM Registers, looks at the DIR register, and examines the MSB and LSB bits of the mask register. This generates the next address. At this point, the FDL may increment or decrement the EAD register by adding or subtracting from EAD the value stored in the pitch register (number of words per each line in memory). It may also shift the contents of the mask register right or left. The position

Figure 3-10. GDC Figure Drawing Hardware



of the next pixel to be written determines these decisions. The results are new values in some of the drawing control logic registers and the new value in the mask register.

The mask register can also be loaded with the MASK command. In this mode, all 16 bits can be set to any desired value. This feature is mostly used for clearing or painting a large area of the memory. In such a case, all 16 bits in the mask register are set to one (FFFFH). By setting figure-drawing parameters or area fill and setting the appropriate direction, all 16 bits are written in each RMW cycle. Since both LSB and MSB of the mask register are always set, in each RMW cycle the EAD is incremented or decremented either by 1 or by pitch value, depending on the direction register.

The character mode also needs some special explanation. There are two ways to write characters and their attributes into video memory. One way is by using FIGS and FIGD commands. The character code is written into RAM locations RA8 and RA9. Mask register is set to all ones or some other masking pattern. The data flows from the RAM to pattern register and is written into video memory with FIGD command. In this way, the same character could be written several times.

Another method uses the WDAT command. In this mode, the data passes directly from the FIFO into the pattern register and then is used by the RMW cycle. The WDAT command is issued only once and all the characters are passed as attributes, with the address automatically incremented after each 2 bytes.

PATTERN REGISTER LOGIC

The pattern register is loaded in one of three ways.

Using the PRAM Command. The parameter RAM locations RA8 and RA9 can be loaded with 16-bit data patterns that are used for drawing solid, dotted, dashed, dot-dash, etc lines during the figure drawing process. At the beginning of the drawing process, the RMW cycle counter is pointing to bit zero of MUX4. This bit is sent to all 16 logic units and is used as explained in figure 3-4 through 3-9. The pointer is circulated toward MSB at each RMW cycle so that the pattern repeats itself every 16 bits.

Using the WDAT Command. This is used mostly in character modes. The pattern register is loaded with the character codes and attributes that are used by the RMW LOGIC to write or modify the data in the video memory. The MUX5 is set for character mode and all 16 bits are sent directly to the logic unit.

Using the DMAW Command. The data to be transferred to the video memory is loaded into the pattern register either as a high byte, a low byte, or a 16-bit word. During the RMW cycle, either 8 or 16 bits are used to modify the read data from video memory to be written back.

COMMAND PROCESSING SPEED

Table 3-1 will give the programmer an approximation of the time it takes the GDC to process command and parameter bytes before the execution of a command. Time is expressed as the number of clock cycles from the time the byte is read by the GDC from the FIFO to the time of reading the next byte or to the start of command execution.

Table 3-1. Processing Times for Command and Parameter Bytes

Command Command Name	Byte Processing Time (in 2xWCLK Cycles)	Parameter Byte Processing Time (in 2xWCLK Cycles)
SYNC or RESET	6	2
VSYNC	12	
START	12	
BLANK	6	
ZOOM	10	2
PRAM	10	4
CCHAR	10	2
PITCH	10	2
LPRD	12	
FIGS	10	2
FIGD	18	
GCHRD	16	
CURS	6	P1, P2: 2 P3: 4 to 64
CURD	14	
MASK	10	2
WDAT (Word)	12	P1: 2 P2: 4
WDAT (High byte)	12	8
WDAT (Low byte)	14	8
DMAW	12	8
RDAT (Word)	14	
RDAT (High byte)	12	
RDAT (Low byte)	14	
DMAR	14	8

The GDC has the capability to draw a number of figure types in the display memory automatically under simple commands from the local microprocessor. To accomplish this, the GDC has internal hardware specifically designed to implement the figure drawing algorithms. A major consideration in the design of the GDC was the speed with which it would calculate the addresses to be modified in display memory during figure drawing. The design implemented in the GDC requires only four clock cycles per read-modify-write (RMW) cycle to draw each pixel. This allows a 100-pixel figure—a vector, rectangle, arc, or graphics character—to be drawn in 400 clock cycles. With a 5-MHz clock, this drawing operation takes only 80 microseconds.

To achieve this high drawing speed, the hardware involved in controlling the drawing process consists of a number of adders, shift registers, and counters organized as a digital differential analyzer (DDA). The command parameters that describe the figure must be initially prepared by the host microprocessor and loaded into the GDC. From this point on, the GDC needs no further assistance from the microprocessor to draw the specified figure. The parameters for the next figure can be determined and loaded while the GDC is drawing, so that the highest system throughput is achieved.

DOT ADDRESSING

During figure drawing, individual dots (pixels) in the display video memory must be modified to form the desired figure. The cursor specify command, CURS, allows the figure's initial dot address, dAD, to be specified along with the address of the word, EAD, in which it resides. Internally, this 4-bit binary dot address, dAD, is expanded to a one-of-sixteen value, which is placed in the mask register. Modification of the data read from display memory during RMW cycles will take place only in the bit positions in which there is a one in the mask register. The one-of-sixteen pattern ensures that only the one proper bit will be modified during figure drawing RMW cycles. The other 15 bits will be written back into display memory without change.

During figure drawing, the GDC makes assumptions about the way the display memory is structured and displayed. The first is that the width of the display memory word must be 16 bits, so that the figures will be drawn correctly. Second, the least significant bit (LSB) of each word must be the first pixel to be serialized and sent out to the display. The most significant bit (MSB) of each word must be the last pixel to go out to the CRT. In this way, the commonly accepted bit numbering scheme where the LSB equals bit number 0, lets the bit number equal its pixel address on the screen when counted from left to right. In other words, the left-most displayed pixel of each word is the LSB of the memory word, contrary to the way memory words are generally represented, with the LSB (bit 0) shown as the right-most bit.

The GDC includes hardware to rotate the mask register right and left to point to the next pixel of the figure. When the extreme right or left bit position is reached, the word address, EAD, is adjusted to continue the drawing process in the neighboring word. As a figure passes through a word "vertically," the mask register carries the proper dot address into the next word.

The mask register may be set to other than a one-of-sixteen pattern with the mask command. This capability is used mainly when the GDC is in character mode but can also be used in graphics mode. Modified figure drawing can be done with "all ones" in the mask register. In this case, the drawing control logic will always detect a one in the appropriate extreme bit position no matter how many shifts have been done. Therefore, the EAD will always be advanced to the next word address. Also, all the bits of the display memory word will be modified because the mask register bits are all set. The effect will be to draw the figure "by the word" instead of "by the pixel" as would be normal. A mask register pattern other than "all ones" or one-of-sixteen will draw meaningless patterns.

DISPLAY MEMORY

Overview

In order to prepare figure drawing commands for the GDC, a programmer must have a thorough understanding of the display memory architecture, including size and addressing structure. Actual correspondence with the displayed image must be clear. Additionally, a number of terms must be unambiguously defined.

Modes of Operation

Graphics figures can be drawn automatically by the GDC only when the display memory, where the figure is to be drawn, is treated as a bit-mapped graphics area. Figure drawing in a character code area will not generate standard graphics figures. In the graphics-only mode, 18 address bits are available for addressing up to 256K words of display memory (4 megapixels). Each word contains 16 horizontally adjacent pixels. In the mixed graphics and characters mode, 2 address bits are used for other purposes, so that only 16 address bits are available for addressing up to 64K words (1 megapixel). In this mixed mode, care must be taken to avoid drawing figures into the character area of display memory, since both areas can be in use at the same time in the same display memory.

Linear Address Space Concept

The display memory is organized like a standard computer program memory: from the first location in memory to the last, there are no discontinuities or missing memory locations. As differentiated from an X,Y coordinate addressed memory, this linear address space is easy to visualize and very flexible. The GDC scans this linear, one-dimensional memory to generate an X,Y, two-dimensional display on the CRT without any need for actual two-dimensional addressing (line and pixel number) on the display memory.

Standard computer programming uses a similar technique to represent a multidimensional array of values. Imagine a two-dimensional array DM with 9 entries arranged as three columns and three rows as in figure 4-1.

The subscripts of the entry labels are in ordered pairs representing first the column of the particular entry and then its row (X coordinate, Y coordinate). Note that the directions in which the row and column numbers increase put the origin (0,0) in the upper left corner. This is similar to the fourth quadrant of the Cartesian plane (figure 4-2).

This array can be stored in a conventional linear memory only if its rows (or columns) are laid end to end in memory as in figure 4-3.

Figure 4-1. Two-Dimensional Array, DM

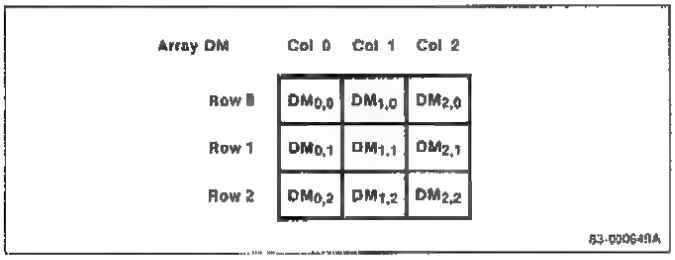
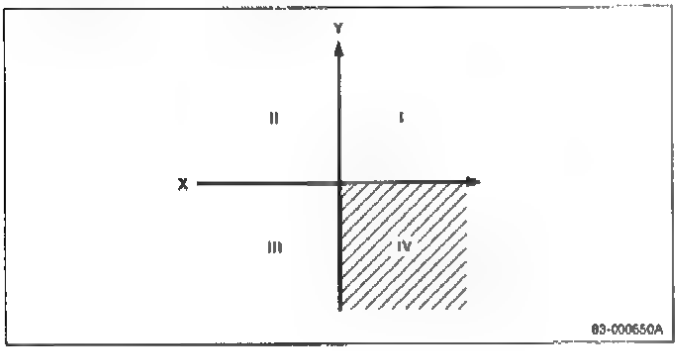


Figure 4-2. Cartesian Plane



This technique has a major advantage. To change to an array of a different size or dimension, no change to the memory structure is needed as long as the memory is large enough to hold all of the array entries.

Going back to the original array, we can enter the actual memory addresses in place of the ordered pairs as in figure 4-4.

Since this array is stored by rows, the number of entries in the array along that dimension (P or pitch) is of fundamental importance. Any horizontal moves are accomplished with simple increments and decrements. Vertical moves require the addition or subtraction of P, while diagonal moves require a combination of both of these operations.

The GDC stores the two-dimensional array, the image to be displayed on the CRT, in its linear display memory in a fashion similar to our array example.

By choosing the quadrant of the numbering scheme to coincide with the raster-scanning process of the CRT, and using row (line) end-to-end storage, linear display memory is easily scanned to generate a video raster on the CRT.

Display Memory Contents

Because the GDC addresses the display through a 16-bit data bus, each word is interpreted differently depending on whether it is defined as a graphics or character area. In the graphics mode, the 16 bits are used for 16 horizontally adjacent picture elements (pixels) which are serially output to the CRT. In the character mode, some of the 16 bits are sent through a character generator, where they are converted into the

Figure 4-3. Conventional Linear Memory

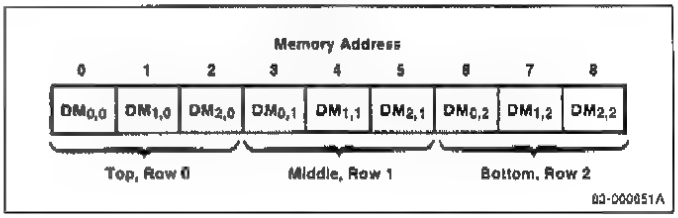
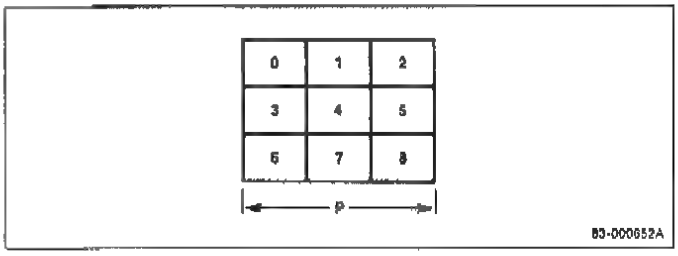


Figure 4-4. Array DM



separate lines of pixels that make up the various characters. They are then scanned out by repeatedly scanning the same row of display memory words while adding the lines.

This line count is fed to the character generator to select which line of the character is to be displayed. Bits not sent through the character generator may be used to select various attributes that may affect the appearance of the displayed character. Typical attributes include blinking, underline, half intensity, color, etc. These attributes are decoded externally to the GDC so that they are under the user's control.

For graphics display areas, the GDC assumes that the 16 bits of each word are serially output to the CRT display in a particular order. The least significant bit, 0, is to be sent out first. The figure-drawing algorithm uses this premise. See figures 4-5 and 4-6.

Note that the bit number increases toward the right, in the direction of increasing word address.

PIXEL ADDRESSES

EAD and dAD Definitions

There can be up to 2^{22} (4M where M = 1,048,576 = 2^{20}) pixels in the display memory, organized into 2^{18} (256K) 16-bit words. The address of one of these pixels is specified to the GDC in two parts. First, an 18-bit address selects the display memory word that contains the pixel. Second, a 4-bit value points to the individual pixel within the word. The word address is called the execute address, or EAD, and effectively acts like a cursor in the GDC. The pixel address is called the dAD, or dot address.

Figure 4-5. Word in Display Memory

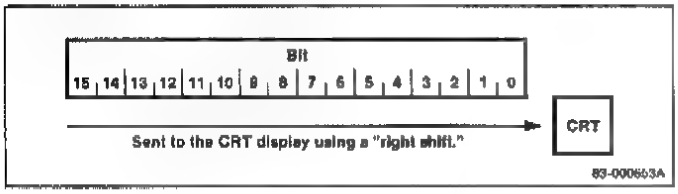
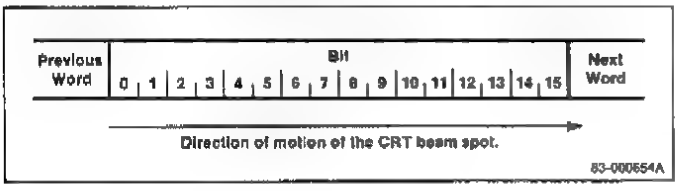


Figure 4-6. Word Appearance on CRT



X, Y to Memory Address Conversion

Since it is likely that many GDC applications will maintain the vector list using X, Y representation, an example of the conversion from these X, Y coordinates to the linear display memory address is included. For simplicity, the origin (0,0) of our display will be in the upper-left corner. This is the same point as the start of the CRT raster-scan. The commonly used origin position at the bottom left of the screen can be used with only a simple substitution in the equations we will develop. To get a clear idea of the conversion from X, Y address to linear address, refer to figure 4-7 and the definitions below.

(XMAX + 1) is a multiple of 16 pixels up to 4096. (YMAX+1) is any number (up to a total of 4 mega-pixels on display memory).

The actual displayed image can be no larger than the display memory structure and must have an even multiple of 16 pixels in the horizontal direction, and no more than 1024 lines vertically (2048 interlaced display).

Figure 4-8, showing a close-up view of the upper-left corner, reveals the relationship between dots and words as seen on the CRT display screen.

Figure 4-7. Display Memory Shown as It Displayed

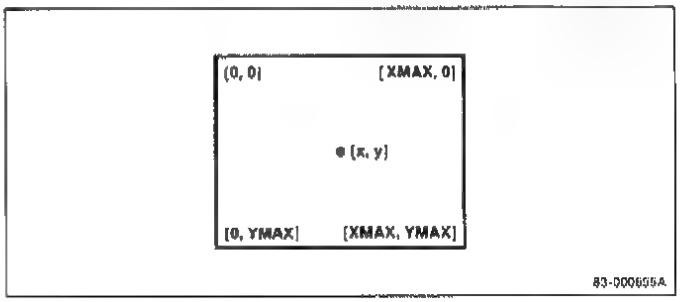
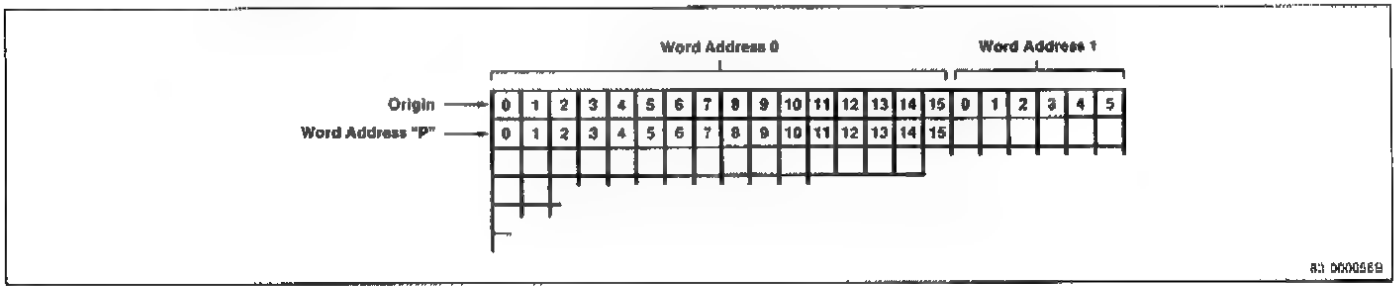


Figure 4-8. Addresses of Dots Within Their Respective Words



A more global viewpoint in figure 4-9 shows the arrangement of word addresses:

P = Number of 16-bit words across one line
P = (XMAX + 1)/16

Finding the Word Addresses. In figure 4-10, note the direction of increasing x and y values. Beginning with the Y coordinate, Y, at the point for which we need to find the absolute memory address, we can find the base address starting at the line in which the pixel lies:

Line Base Address (LBA) = P (Y)

To this we must add the number of words along the line due to the value of the X coordinate to find the actual EAD value:

EAD = LBA + INTEGER (X/16)
= P(Y) + INTEGER (X/16)

where the integer operation yields the integer part of the X/16 division, truncating off the fractional part or remainder of the division.

Figure 4-9. Word Addresses on Display Memory

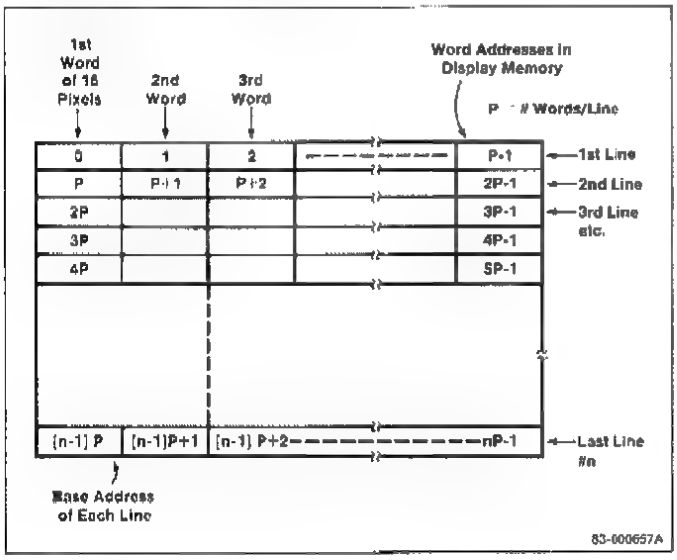
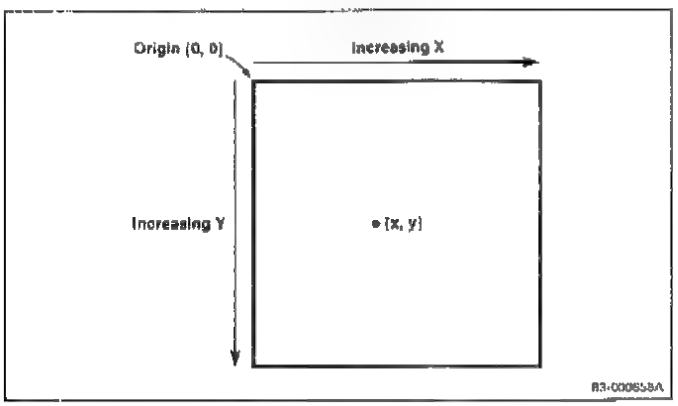


Figure 4-10. Direction of Increasing X,Y Values



Finding the Dot Addresses. To find the dot addresses within the word determined in the EAD equation, take the residue or remainder of the division performed above and treat it as the integer value of the dot address:

dAD = REMAINDER (X/16) x 16
= RESIDUE (X/16)

Here's an example using actual values. Let the display memory be configured as 512 pixels by 512 lines (XMAX = YMAX = 511), and the point will be (X, Y) = (231, 475) (All numbers use base 10 notation).

P = (XMAX + 1)/16 = 512/16 = 32 Words/Line
EAD = PCY + INTEGER (X/16)
= 32(475) + INTEGER (231/16)
= 15,200 + INTEGER (14.43750)
= 15,214

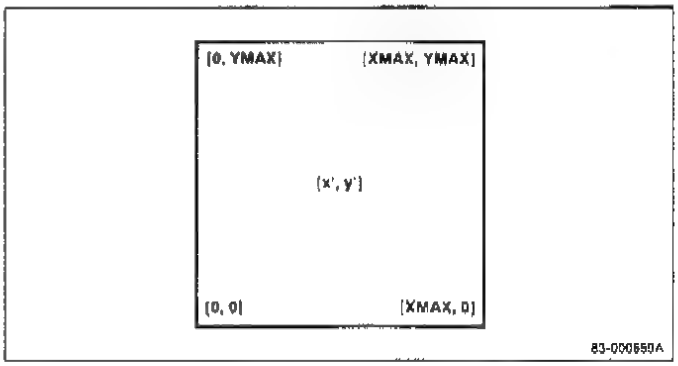
dAD = RESIDUE (X/16) = RESIDUE (14.43750)
= 16 x 0.4375 = 7

These two numbers (converted to base 2) can be sent to the GDC to specify the particular pixel of interest.

Calculation Considerations. Note that all the values are unsigned positive integers. Certainly, this is the easiest format with which to work. In addition, with the possible exception of the multiplication by P in the dAD calculation, all the multiplication and division is accomplished by simple shifts and done by powers of two. If the display memory is a "power of two" pixels wide, the P will be also. Its multiplication will be just arithmetic shifts. If memory is structured with other than a "power of two" width, a real multiply instruction will be needed. If the processor doing these calculations does not have a multiply instruction, the "shift and add" multiply routine is neither difficult nor time-consuming to write.

Other Coordinate Systems. Setting the origin at another point on the screen does not seriously complicate the physical memory address calculation. Imagine a system with a lower-left-corner origin, denoted by primed coordinates (figure 4-11).

Figure 4-11. Origin at Lower Left



A simple coordinate translation relationship can be defined between these two systems:

$$X = X'$$
$$Y = YMAX - Y'$$

With these expressions substituted into the EAD and dAD equations, only one difference is noted:

$$EAD = P\ (YMAX - Y') + INTEGER\ (X'/16)$$
$$dAD = RESIDUE\ (X'/16)$$

Any other coordinate system can be similarly defined.

Display Memory Organization. Let's examine the display memory organization before explaining how to calculate the parameters for figure drawing.

The GDC in the graphic display mode provides 18 address lines, 16 lines of these also serving as a bidirectional data bus. This interface to the display memory allows direct control of all 262,144 or 256K (where K = 1024) addresses, each containing 16 bits of data for a total of 4096K bits of memory.

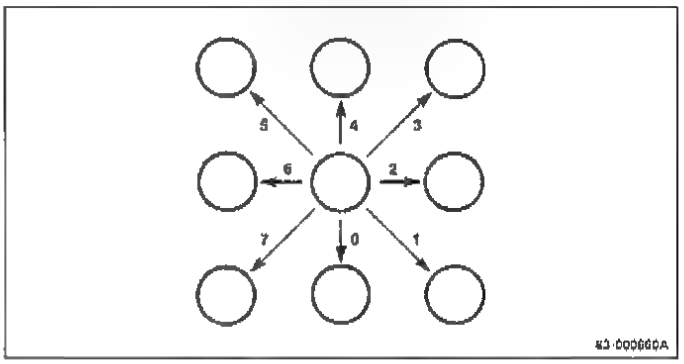
If each bit of display memory corresponds to one pixel on the display screen (for a black and white monitor), this allows the system to be configured in several ways; for example, 2048 x 2048, 4096 x 1024, or any other combination. Since today's CRT monitor technology does not generally allow resolutions this high without prohibitively high costs, in most applications the display area will be of a smaller size, such as 512 x 512, 768 x 576, etc. In this instance, the display memory contains many pages of the display area or it can be used for displaying a smaller window of a much larger image.

Design of a color graphics system needs special consideration. The most common way of implementing color is by building multiple planes of the display memory, each plane representing one of the primary colors. By sending the video data from all the planes together, a large number of colors may be generated. For example, with three planes (red, blue, and green) a total of six colors, plus black and white, may be generated. A similar technique can be used for generating monochrome grayscale displays.

DRAWING DIRECTIONS

Every figure drawn by the GDC consists of a large number of individual bits that must be modified in the display memory. As figure drawing proceeds, the next pixel to be modified can be any one of the eight nearest neighbors of the current figure pixel. The figure drawing hardware decides which direction to proceed based on the selected drawing algorithm and position within the figure. DIR 0 lies in the direction of the pixel directly below (as it would be seen on the display screen), and the direction number assignments then proceed in a counterclockwise fashion through all eight directions. See figure 4-12.

Figure 4-12. Drawing Directions



Determining the next pixel of the figure during figure drawing requires the proper manipulation of the word address and the pixel bit position according to the drawing direction. To move to the pixel below or above, it is necessary to add or subtract the number of memory words per line to or from the word address pointer, EAD. This offset is the pitch, P, of the display memory. To move to a pixel to the right or left across a word boundary, the EAD is incremented or decremented and the mask register is shifted in the proper direction with end wraparound. For diagonal directions, the add/subtract of the P operation must be combined with mask register shifting. Table 4-1 summarizes these operations for each direction.

Table 4-1. Operations to Address the Nest Pixel

DIR	Operations to address the next pixel
0 (000)	EAD + P → EAD
1 (001)	EAD + P → EAD dAD (MSB) = 1:EAD + 1 → EAD dAD → LR
2 (010)	dAD (MSB) = 1:EAD + 1 → EAD dAD → LR
3 (011)	EAD − P → EAD dAD (MSB) = 1:EAD + 1 → EAD dAD → LR
4 (100)	EAD − P → EAD
5 (101)	EAD − P → EAD dAD (LSB) = 1:EAD − 1 → EAD dAD → RR
6 (110)	dAD (LSB) = 1:EAD − 1 → EAD dAD → RR
7 (111)	EAD + P → EAD dAD (LSB) = 1:EAD − 1 → EAD dAD → RR

Where: P = pitch, LR = left rotate, RR = right rotate.
EAD = Execute word address and
dAD = Dot address stored in the mask register.

When a figure is being specified to the GDC for figure drawing, the initial direction, DIR, must be supplied. The various figures are thereby given their basic rotational orientation. As drawing progresses, the drawing direction is adjusted by the GDC for each pixel, where necessary, so that pixels in the right positions are modified to draw the figure.

INITIAL DRAWING DIRECTION

For the various figures, the effect of the initial direction specification upon the resulting drawing is shown in table 4-2. For each of the eight initially specified directions, the various figure orientations are shown. The heavy dot in each of the drawings in the table represents the starting point of the figure drawing process.

For lines and vectors, table 4-2 shows the range of line directions which can be handled within each initially specified direction octant. Note that a line to be drawn along the counterclockwise boundary of each octant should be drawn using the next higher DIR value.

For drawing arcs, the starting point in each octant is on its coordinate axis. See figure 4-13. The arc is then drawn toward the octant's diagonal boundary. The tangential arrow in table 4-2 shows the initially specified direction. The iterative nature of the digital differential analyzer's operation requires that it calculate all the pixel addresses from a starting point on the axis to the end of the arc. Any pixels between the start of the arc and the axis are calculated but not drawn into the display memory.

Table 4-2. Figure Orientations.

Dir	Line	Arc	Character	Slant Char.	Rectangle	DMA
000						
001						
010						
011						
100						
101						
110						
111						

83-000661A

The specifications for these arbitrary arc lengths are computed by the host computer using two angles measured from the octant's coordinate axis. See figure 4-14. The angle measured to the far end of the arc is used to compute the total number of pixels. The angle to the start of the arc yields the number of pixels not to be included in the figure. Both of these numbers are given to the GDC as parameters in the FIGS command. Arcs which cross the boundaries of octants must be broken down into arc segments, each constrained to one octant, see figure 4-15. Circles are drawn using eight full 45-degree arcs.

Figure 4-13 Arc Drawing Directions

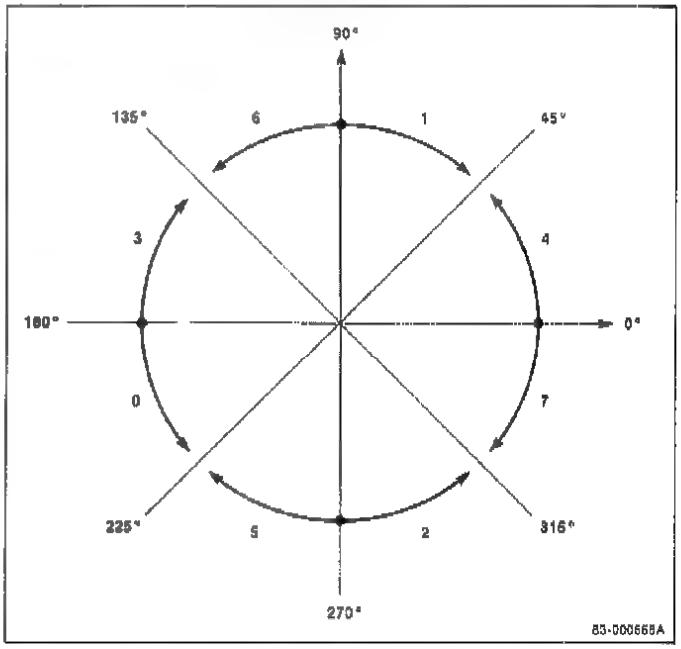


Figure 4-14. Arc Drawing Within an Octant

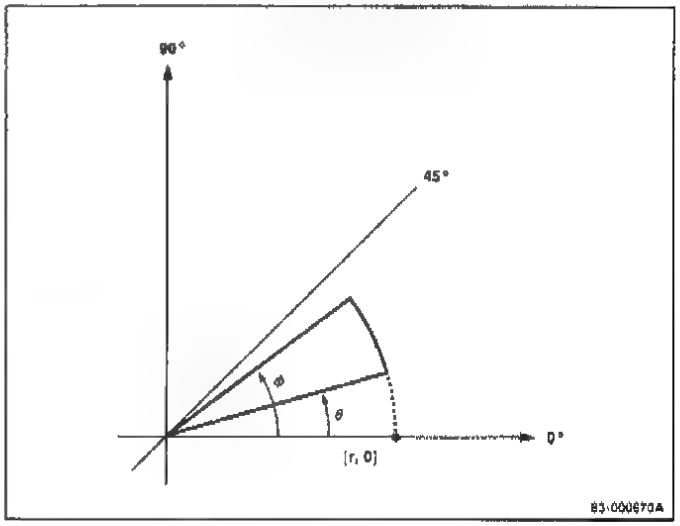
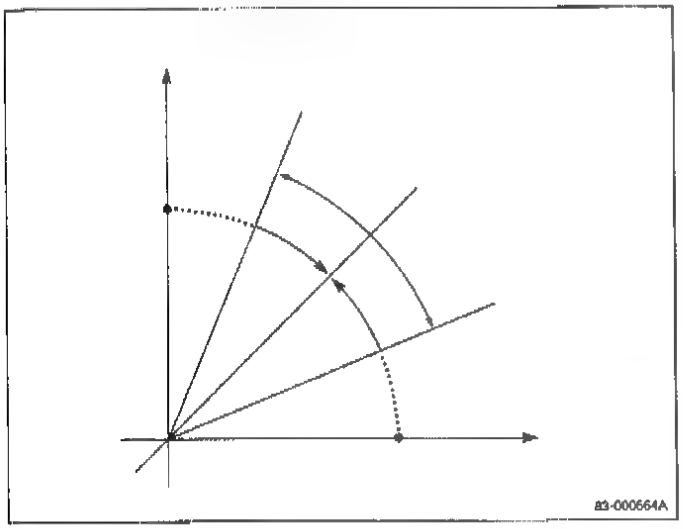


Figure 4-15. Arc Drawing Across Octant Boundaries



For vectors and arcs, the slope and radius of curvature information within the octant is specified by other parameters of the figure specify command, FIGS, as will be discussed below.

To draw graphics characters, the contents of the GDC's parameter RAM is moved into the bit-map display memory one pixel at a time. The parameter RAM is scanned in the same order for any of the drawing directions so that the drawn characters appear merely to be rotated around the starting point. Although the parameter RAM holds 64 bits in an eight-by-eight array, the pattern drawn can be made smaller or larger than this by supplying the appropriate FIGS parameters. If larger, the pattern drawn into display memory is a repetition of the pattern in the parameter RAM in both the x and y directions. This drawing mode can be used to fill an area in the display memory with solid or pattern, in addition to drawing bit-mapped characters.

Slanted graphics characters and area-filling patterns are drawn in a fashion similar to graphics characters, except that each row of pixels in the character is offset by one pixel position sideways from the previous row. The result is a 45-degree slanting effect similar to italics printing. Slanted graphics characters can also be drawn in all of the octant directions, in each case maintaining the slanted appearance.

For rectangle drawing, the GDC is given the pixel counts for the two adjacent sides of the rectangle and one of eight initial directions. As with graphics characters, 45-degree rotated rectangles are generated with odd DIR values. The sides of the rectangle can be up to 16K pixels long per side.

DMA transfers are handled on a word basis instead of a pixel basis. In this way, very-high-speed transfers can be made to and from solid rectangular areas of display memory. Table 4-2 shows the starting word address in display memory with a heavy dot for each of the drawing directions. The next word to be accessed is found in the direction of the arrow. The specified number of words is transferred using the initially specified direction, and then the next group is accessed, repeating until the specified number of groups is transferred. The 45-degree rotated patterns advance the word address both vertically and horizontally as the arrow paths are followed.

THE DIGITAL DIFFERENTIAL ANALYZER

The digital differential analyzer, DDA, in the GDC generates figures from their differential equations with dedicated hardware at very-high speed. The DDA works in parallel with the Read-Modify-Write (RMW) hardware to form a pipelined system for figure drawing. Both lines and circles can be drawn, as well as derived forms such as rectangles, graphics characters, and DMA transfers. The figures are drawn into display memory using back-to-back RMW drawing cycles while the DDA is calculating the address of the next pixel of the figure.

The analyzer used in the GDC uses one coordinate axis direction (either the X or Y axis) as the independent axis of the figure in that octant. The pixel count along this direction is equal to the total number of pixels in that figure segment, because the analyzer advances along this axis for each pixel that is drawn. The other axis direction is the dependent axis, and the pixel distance in this direction is always less than or equal to the distance along the independent axis.

As each pixel is drawn, the dependent axis may or may not be advanced, depending on the figure type and the position within the figure. The result is that the figure's pixels are always within one half-pixel of the mathematically ideal position of the figure. The independent axis advance can occur in the +X, −X, +Y, or −Y direction, depending on the DIR specified octant. The host computer must determine the appropriate parameters for the desired figure in these terms and use the DIR parameter to provide the correlation to the actual figure's orientation.

FIGURE DRAWING

Once all the commands and parameters have been interpreted by the GDC's command processor and the drawing process begins, the digital differential analyzer and the RMW hardware work together to do the figure drawing (or DMA transfers, etc). They are designed to work in parallel with each other, so that

while the RMW logic is modifying a pixel in the figure, the DDA is determining the address of the next pixel. Thus, drawing proceeds at the fastest rate possible for this hardware.

There are two levels in this pipeline: the DDA and the RMW logic. If no cycles are to be wasted, the first pixel address must be supplied by a source outside the DDA so that the RMW logic can have something to modify during the first cycle. Of course, this is no problem, since the host microprocessor must supply the address of the first pixel anyway (cursor position). It is important to note that the dot count (really an address calculation cycle count) provided to the DDA does not include this first cycle which draws the first pixel. Also interesting to note is that when the dot count expires and while the last pixel is being written into display memory, the DDA calculates the address of the pixel which would have been drawn next if the dot count had been greater and leaves the cursor values, EAD and dAD, pointing at this pixel.

PREPARATION

To prepare the GDC to draw a figure, the details of the desired drawing operation must be loaded into the GDC through the command and parameter FIFO. The type of RMW cycle modification can be selected using the WDAT command followed by no parameters. The cursor can be positioned using the CURS command to the word and dot address, EAD and dAD, of the starting pixel of the figure. Parameter RAM bytes 8 and 9 must be loaded with the drawing pattern for continuous, dotted, dashed, etc figure lines using the PRAM command. For graphics character drawing and patterned area filling, parameter RAM bytes 8 through 15 can be loaded with the desired pattern or character. In either case, the figure specify command, FIGS, is used to set up the digital differential analyzer, DDA, for the desired figure, and a figure drawing command, FIGD or GCHRD, is issued to begin the actual drawing process.

During the read-modify-write memory cycles needed to draw the desired figure into display memory, the type of logical operation performed can be selected from among the following: SET, CLEAR, COMPLEMENT, and REPLACE. The WDAT command can be used to configure the GDC for the desired operation. The TYPE field (command byte bits 3 and 4) should select word transfers (00 code), and command bits 0 and 1, then select the logical operation. See tables 4-3 and 4-4.

The WDAT command byte must not be followed by any parameter bytes when only used to set the RMW modification operation. If parameters were to follow, the GDC would move them into display memory as it would if a block data transfer were intended. The next command byte will terminate the WDAT command-byte-only sequence without any parameters.

Parameter RAM Bytes 8 and 9 must contain the pattern to be drawn as the figure. As figure drawing begins, the contents of these RAM bytes will be moved into the pattern register and used by the RMW hardware to enable or disable the modification of the individual pixel bits of the figure as their addresses are calculated during drawing. In graphics mode, the pattern register bits are used one at a time as selected by a counter which increments after each RMW cycle. The LSB of the pattern is the first bit to be used for the patterning. The second RMW cycle uses the second bit, etc. The pattern impressed on the figure is repeated every 16 RMW cycles. Because any 16-bit pattern can be put in the parameter RAM before drawing, any modulo-16 pattern may be generated along the figure. The pattern register is always scanned in the same sequence, regardless of the type or direction of the figure being drawn.

Table 4-3. WDAT Command Format

7	6	5	4	3	2	1	0
0	0	1	Type-1	Type-0	0	Mod-1	Mod-0

Table 4-4. RMW Logical Operations

Bits 1 and 0		Operation
0	0	REPLACE the addressed pixel bit with the value in the Pattern register's selected bit position.
0	1	COMPLEMENT the addressed pixel bit if the selected bit in the Patern register is a ONE.
1	0	RESET the addressed pixel bit to a ZERO if the selected bit in the Pattern register is a ONE.
1	1	SET the addressed pixel bit ot a ONE if the selected bit in the Pattern register is a ONE.

FIGS PARAMETERS

The figure specify (FIGS) command's parameters control the figure to be drawn. Its first parameter byte configures the DDA for the basic figure type. The rest of the parameter bytes load the different variables to guide the DDA during figure drawing. Each variable is a 14-bit integer which is loaded by the GDC's command processor into the appropriate register using two parameter bytes. Where the parameters can be either positive or negative, two's complement notation should be used. This is an easy requirement to meet when using a microprocessor or minicomputer to drive the GDC, as these machines commonly use two's complement arighmetic internally. The interpretation of these values by the DDA varies with different figure types.

The five figure drawing variables are called DC, D, D2, D1, and DM, and their interpretation by the DDA varies for the various figure types. They are accepted by the GDC in this order only.

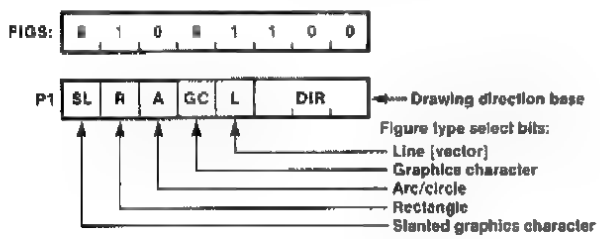
As the GDC's command processor executes each FIGS command opcode, the various DDA registers have specific initial values: DC = 0, D = 8, D2 = 8, DI = -1, and DM = -1. To change these values the appropriate parameter bytes must follow the FIGS command byte. In several figure types, some of these initial values need not be changed and consequently those parameter bytes need not be sent to the GDC. The internal command processor in the GDC accepts the parameters as they come and puts them in the appropriate DDA registers in a fixed sequence. This loading process is terminated by any command byte, such as the figure drawing start command, FIGD. Therefore, it is possible to load only the first two variables, or the first three, etc. It is not possible to load the first, skip the second, and load the third variable, for example.

The first parameter byte describes the type of figure to be drawn. The top five bits are set to indicate the various figure types. Just one of these bits can be set at a time, except for the slant bit, which can be used only in conjunction with the graphics character bit. Use the specific bit combinations listed in table 4-5 exclusively, as only these assure correct drawing operation.

Table 4-5. Figure Type Select Bit Combinations

SL	R	A	GC	L	Operation
0	0	0	0	0	Character display mode drawing, individual dot drawing, DMA, WDAT, and RDAT
0	0	0	0	1	Straight line drawing
0	0	0	1	0	Graphics character drawing and area filling with graphics character pattern
0	0	1	0	0	Arc and circle drawing
0	1	0	0	0	Rectangle drawing
1	0	0	1	0	Slanted graphics character drawing and slanted area filling

FIGS Parameter Byte 1



The DIR bits specify the initial drawing direction to the DDA to control the orientation of the resulting figure. Typically the figure type select bits are known before any figure drawing parameters are calculated for each figure, as the first step in determining the other DDA parameters.

FIGS COMMAND-TYPE PARAMETER

The figure specify GDC command, FIGS, sets up the GDC's digital differential analyzer (DDA) to perform one of a number of read-modify-write (RMW) operations. The DDA can generate six different types of operations in any of eight octants, with wide parametric variations. After the FIGS command has prepared the DDA to do the proper address calculations, the RMW operation is initiated with one of several RMW operation start command opcodes.

The five figure type bits of the first FIGS parameter byte (table 4-5) configure the DDA for the general type of operation to be performed. The other field of the FIGS parameters and the particular invocation command given further specify the operation to be performed within this figure type. There are just six valid figure types which can be selected. Only these specified operations assure correct drawing operation.

RMW Operation Start Commands

After the FIGS command and its parameters are sent to the GDC, the host processor must initiate the RMW operation with one of the following commands:

FIGD	=	Figure draw start
GCHRD	=	Area filling and graphics character drawing start
WDAT	=	Write data into the display memory
RDAT	=	Read data from the display memory
DMAW	=	DMA write sequence initiate
DMAR	=	DMA read sequence initiate

Each of these start commands can be used with one or several figure types.

Correlation Between Command and Figure Type

It is informative to organize the various figure type options by the start command that will call them into operation. This is done in table 4-6. Note that area filling is the operation used for graphics character drawing.

Table 4-6. Allowed RMW Operations

Command	Allowed RMW Operations	SL	R	A	GC	L
FIGD	Line or vector drawing	0	0	0	0	1
	Arc and circle drawing	0	0	1	0	0
	Rectangle drawing	0	1	0	0	0
	Single dot drawing	0	0	0	0	0
GCHRD	Area filling	0	0	0	1	0
	Slanted area filling	1	0	0	1	0
WDAT	Single word	0	0	0	0	0
	Successive word writes	0	0	0	0	0
	Character mode reading	0	0	0	0	0
RDAT	Single word read	0	0	0	0	0
	Successive word reads	0	0	0	0	0
	Character mode reading	0	0	0	0	0
DMAW	DMA write sequence	0	0	0	0	0
DMAR	DMA read sequence	0	0	0	0	0

VECTOR DRAWING

Vector (line) drawing requires nine parameter bytes following the FIGS command byte. The first parameter byte must have the L bit set and the drawing direction octant specified. Four pairs of parameter bytes must follow to be loaded into the GDC's digital differential analyzer (DDA) registers: DC, D, D2, and D1. These values must be determined by the host microcomputer based on the desired vector's starting and ending points.

For vector drawing, the GDC draws with a technique based on independent and dependent axes. The host microcomputer must determine which of the X or Y axes is the independent axis and which is the dependent axis for the octant direction in which the desired vector lies. This can be determined by an examination of the vector's end point coordinates. The results can then be used to prepare the required figure drawing parameters rapidly.

The first step in this process is to calculate the number of pixel positions spanned by the vector along each axis:

$$\text{DeltaX} = \text{EndingX} - \text{StartingX}$$
$$\text{DeltaY} = \text{EndingY} - \text{StartingY}$$

The starting values are the x, y coordinates of the vector's starting point and the ending values are the coordinates of the vector's ending point. The delta values can be either positive or negative.

Note that the delta values equal the number of pixel distances along each axis but do not equal the number of pixels to be drawn. They are one pixel short of the number of pixels needed to cover the distance because the first pixel is not taken into account in these equations. The GDC draws the pixel pointed to by the cursor during its first drawing cycle, while the DDA is calculating the address of the second pixel. The dot count passed to the DDA as the DC parameter refers to the number of pixel addresses to calculate and draw, after the first pixel is drawn. While the last pixel is being drawn (the DC count has reached zero), the cursor is advanced to the point that would have been the next pixel of the figure if the DC value had been larger.

In figure 4-16, note that the Y-axis positive direction is downward as the display memory is viewed on the CRT display. This reflects the upper left corner position of location 0 in the display memory. Other coordinate systems with different origin positions can be easily converted into this format.

The first step toward determining the direction octant of the vector (figure 4-17) is to find its direction quadrant. This is easily determined by testing the sign of the two delta values:

DeltaX Sign	DeltaY Sign	Quadrant
Positive	Negative	1
Negative	Negative	2
Negative	Positive	3
Positive	Positive	4

Figure 4-16. Direction Quadrant Definition

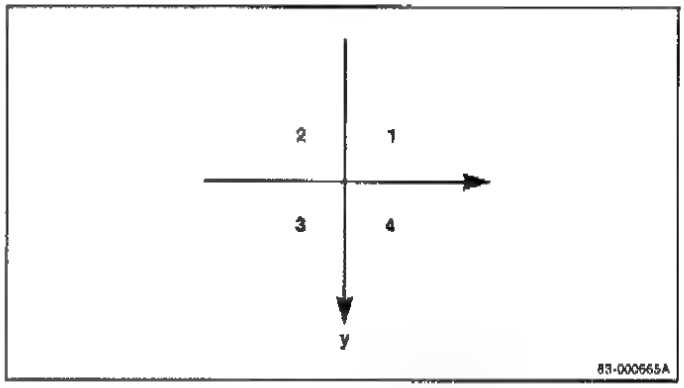
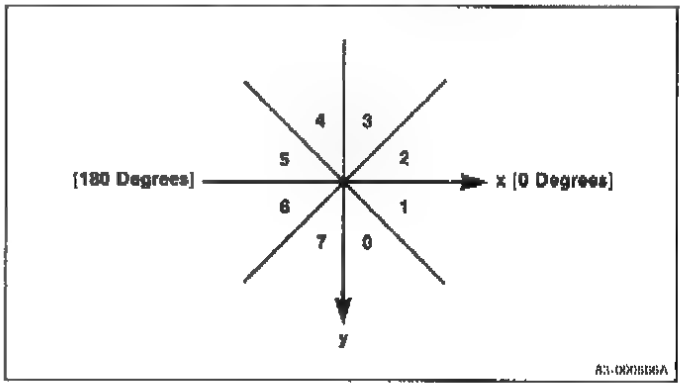


Figure 4-17. Octant Direction Definition



Within the quadrant, the direction octant may be determined by comparing the absolute value magnitudes of the two delta values. If ABS (DeltaX) is greater than ABS (DeltaY):

Quadrant	Octant
1	2
2	5
3	6
4	1

If ABS (DeltaX) is less than ABS (DeltaY):

Quadrant	Octant
1	3
2	4
3	7
4	0

If ABS (DeltaX) = ABS (DeltaY), the vector is on the diagonal octant boundary within the quadrant. The higher-numbered, odd octant of the quadrant can be chosen in this case to maintain symmetry with vector direction assignments along the coordinate axes.

These tests can be made at high speed if the signs of the two delta terms and their difference are used to index into a software table.

With the vector's direction determined to be within a particular octant, the next step is the determination of the independent drawing axis and calculation of the drawing parameters. The independent axis will be the X-axis for those octants where ABS (DeltaX) is greater than ABS (DeltaY), and it will be the y-axis for those octants where ABS (DeltaX) is less than or equal to ABS (DeltaY). As can be seen from figure 4-17 and the tabulation below, the Y-axis is the independent axis for those octants where the line is longer in the y direction than it is in the X direction.

Octant	Independent Axis	Dependent Axis
0	Y	X
1	X	Y
2	X	Y
3	Y	X
4	Y	X
5	X	Y
6	X	Y
7	Y	X

The GDC's DDA drawing parameters can be calculated from the absolute values of DeltaX and DeltaY. The octant drawing direction parameter, DIR, directs the DDA to advance along the independent axis by incrementing or decrementing the x-address or the y-address, depending on which octant direction is specified. Similarly, the manipulation required to advance along the dependent axis is also implicitly specified by the DIR parameter.

In the line drawing parameter equations below, I is the independent axis and D is the dependent axis.

$$DC = ABS(DeltaI)$$
$$D = [\,2 \times ABS(DeltaD)\,] - ABS(DeltaI)$$
$$D2 = 2 \times [\,ABS(DeltaD) - ABS(DeltaI)\,]$$
$$D1 = 2 \times ABS(DeltaD)$$

It will be necessary to substitute the appropriate DeltaX and DeltaY values for the DeltaI and DeltaD terms in these equations as required by the drawing direction octant. The calculations of D and D2 require 16-bit integer, 2's complement arithmetic. The values sent to the GDC should include only the low 14 bits. Both the DC and D1 values will be unsigned integers with a range of 0 to $(2^{14} - 1)$. Note that the DC parameter contains the pixel (dot) count for the figure minus one, since the count does not include the first pixel. The other three parameters encode the slope of the line as values to be used by the DDA to determine when to advance along the dependent axis as vector drawing proceeds.

VECTOR DRAWING EXAMPLE

An example of calculating a vector's figure-drawing parameters will help make clear the procedure outlined above. Note that all numbers will be shown as sign/magnitude decimal numbers, when in reality they must be given to the GDC in 2's complement or unsigned integer binary form. The starting address is given to the GDC as an unsigned integer via the CURS command, and it must use the linear addressing technique of the display memory. The example's starting and ending points, expressed as (x, y) ordered pairs, will be:

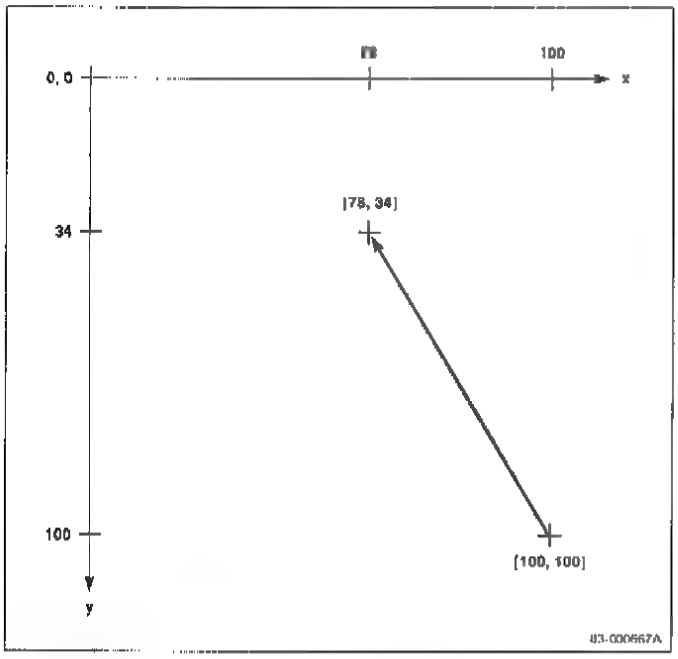
$$\text{Starting Point} = (100,100)$$
$$\text{Ending Point} = (78,34)$$

On the CRT display screen, this vector would appear as shown in figure 4-18.

To determine the direction of the vector, the first step is to calculate the DeltaX and DeltaY values:

$$DeltaX = EndingX - StartingX = 78 - 100$$
$$= -22$$
$$DeltaY = EndingY - StartingY = 34 - 100$$
$$= -66$$

Figure 4-18. Vector Drawing Example



Both of these values are negative; therefore the vector lies in the second quadrant.

To determine the direction octant within the quadrant, we notice that the absolute value of DeltaY is larger than the absolute value of DeltaX, which corresponds to a vector direction in octant 4, so that DIR = 4.

In octant 4, the independent axis is Y and the dependent axis is X. Therefore we can define:

Deltal = DeltaY = −66
DeltaD = DeltaX = −22

The line drawing parameters for the FIGS command can be easily calculated as follows:

DC = ABS (Deltal) = 66
D = [2 x ABS (DeltaD)] −ABS (Deltal)
= (2 x 22) − 66 = −22
D2 = 2 x [ABS (DeltaD) − ABS (Deltal)]
= 2 x [22 − 66] = −88
D1 = 2 x ABS (DeltaD) = 2 x 22 = 44

When expressed in the appropriate binary notation and loaded into the GDC, along with the other set-up commands and parameters, the GDC will draw the desired vector using 67 RMW display memory cycles. Both DC and D1 will always be unsigned integers for vector drawing. They have the full 14-bit range of values from 0 to 16,384 −1. D and D2, on the other hand, must use 2's complement notation for a range of values from 8192 − 1 to −8192.

VECTOR DRAWING ALGORITHM

Let's consider some arbitrary vectors to understand better the parameters that the GDC needs during vector drawing.

The equation of the straight line may be expressed as follows:

$$y_2 = \frac{dy}{dx}(x_2 - x_1) + y_1$$

where x_1, x_2, y_1, and y_2 are coordinates of the starting and ending point of the vector. If we normalize the vector so that the starting point is at (0,0) coordinates, the equation takes this form:

$$y = \frac{ADY}{ADX}(x) \qquad y = \frac{ADY}{ADX}(x)$$

where x and y are coordinates of each consecutive pixel of the vector.

Since in the raster drawing system, x and y values can only be positive integers, it could be expressed as follows:

$$y = \frac{ADY}{ADX}(x) = I + F$$

Where:
x is consecutive X coordinates (1,2,3...),

I is the integer part of Y coordinates (...−2,−1,0,1,2...)

F is a fractional part of Y coordinates ($0 < F < 1$).

The F value of this equation must be rounded off to the nearest integer. In the case where F = 0.5, the I value is incremented by 1 or rounded up. To decide when to round up or round down, the F value must be evaluated by GDC logic at each pixel address. The equation takes the following form to isolate the fractional part:

$$F = \frac{ADY}{ADX}(x) - I$$

Rounding up takes place when F is greater than 0.5:

$$\frac{ADY}{ADX}(x) - I > 0.5$$

The equation achieved by multiplying both sides by two and subtracting 1 is:

$$2\frac{ADY}{ADX}(x) - 2I - 1 > 0$$

The equation achieved by multiplying it by ADX is:

$$2ADY(x) - (2I + 1)ADX > 0$$

The hardware in the GDC uses this equation in calculating each consecutive pixel address. Since it calculates only the next pixel to be drawn at any given moment, the equation could be further simplified.

The first pixel of the vector is drawn at the point specified as a starting point of the vector by the CURS command. At this point x = 0 and I = 0.

For the second pixel, the x value is incremented and I is still equal to zero. That is because ADX > ADY; therefore, the y value will be incremented only if F > 0.5. To evaluate the value of F, equation for D must be evaluated.

$$D = 2ADY\,(x) - (2I + 1)\,ADX$$

for x = 1 and I = 0, D = 2ADY − ADX

Based on the sign of the value D, the next value of the y coordinate (I) is rounded up or down. If D > 0, I is incremented by one. If D < 0, I is not modified. Depending on the sign of the parameter D, the next pixel could be drawn in one of two positions, either above or below the ideal line.

When D < 0, the only parameter between two consecutive pixels that is incremented is the X value (increment is in X direction) and the equation for D1 takes form:

$$\begin{aligned} D1 &= (\text{Next pixel}) - (\text{Present pixel}) \\ &= [2ADY\,(x+1) - (2I+1)\,ADX] \\ &\qquad - [2ADY\,(x) - (2I+1)\,ADX] \\ &= 2ADY \end{aligned}$$

This value is added to value D for the next pixel if D was negative.

When D ≥ 0, both the x and I parameters are incremented and the equation for D2 takes form:

$$\begin{aligned} D2 &= (\text{Next pixel}) - (\text{Present pixel}) \\ &= [2ADY\,(x+1) - (2I+2+1)\,ADX] \\ &\qquad - [2ADY\,(x) - (2I+1)\,ADX] \\ &= 2\,(ADY - ADX) \end{aligned}$$

This is added to the value of D, if D was positive or zero.

The parameters D, D1, and D2 must be supplied to the GDC before the drawing process starts. The only other parameters that are necessary are the total number of pixels to be drawn, excluding the first pixel. This number is always equal to ADX. The seven parameters that the CPU must provide to the GDC are:

EAD = Word address of first pixel
dAD = Address of pixel in the EAD of first pixel
DIR = One of the eight directions
DC = ADX or total number of pixels − 1 to be drawn
D = 2ADY − ADX
D1 = 2ADY
D2 = 2(ADY − ADX)

The last three parameters must be loaded into the GDC using two's complement notation signed integers.

ARC AND CIRCLE DRAWING

The GDC's digital differential analyzer (DDA) can draw arcs subtending up to 45 degrees on one octant. Circles can be drawn using eight arcs, each having the same radius of curvature and center, and each spanning 45 degrees of the total circle. Using the same principle of independent and dependent axes that is used for vector drawing, arc drawing is limited to one octant per invocation, so that the advance of the x and y pixel addresses during drawing can always proceed in the same direction.

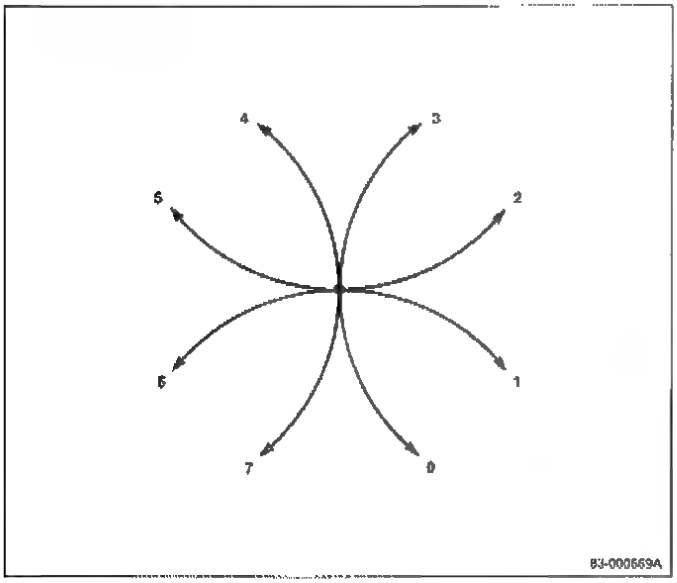
To draw an arc, the GDC is given a starting pixel address with a CURS command, a pattern with a PRAM command load into bytes 8 and 9 (if the previous pattern was different or in doubt), an RMW logical modification operation selection with a WDAT command (if the last specification was different or in doubt), a FIGS command to set up the DDA for the arc, and a FIGD command to initiate the drawing process. The only difference between this and a vector writing command sequence is the FIGS command parameters.

ARC DRAWING CURSOR

The cursor specification command, CURS, is used to position the cursor to the point on the arc where it intercepts the X or Y axis in the particular octant in which the arc is to be drawn. The arc is always drawn in a direction away from this axis and toward the diagonal octant boundary. When circles are drawn, although there are eight arcs required to form the entire circle, there are, therefore, only four starting points, each of which is used for two of the arcs.

Notice in figure 4-13 that there are four initial directions, which can each bend either counterclockwise or clockwise depending on the DIR value specified. The relationship of the various DIR values to drawing direction can be seen more clearly in figure 4-19.

Figure 4-19. Arc Drawing Directions From a Point



Arcs to be drawn across octant boundaries must be drawn in arc segments that do not cross boundaries. If the dot count (DC) given to the DDA in the FIGS command results in the drawing crossing the octant's diagonal boundary, the resulting figure will look like an arc only up to the point where it crosses out of the initial octant. The last three parameters must be loaded into the GDC using two's complement notation signed integers.

ARC FIGS PARAMETERS

The initial values available to the GDC software driver for circle and arc drawing are often the center of the circle (or center of curvature for an arc), the radius, and, for arcs, the initial and final angles subtended as measured from the center of curvature. Using a straightforward example in quadrant 1, the specifications of an arc might be as shown previously in figure 4-14.

In figure 4-14, the initial drawing direction, DIR = 4 (figure 4-19) and the independent axis is Y. Note that a vector drawn by the GDC from the arc's center of curvature will use a DIR = 2 to be drawn in this area of the circle, while the arc uses a direction initially tangential to the radius of the arc so that its initial DIR is at right angles to the vector's. Note that a 90-degree rotation of direction is equivalent to adding or subtracting 2 from the DIR value.

The arc must be computed by the DDA from a starting point on the axis even if the desired arc will not touch the axis. Therefore, a dot count must be specified which includes all the dots between the axis and the far end of the arc. Since the number of pixels (dots) along the independent axis is equal to the total number of pixels calculated, the DC dot count parameter can be calculated using the sine function. The number of pixels along the arc's length is equal to the DeltaY pixel count to the far end of the arc from the axis (in this case, the x-axis).

This DeltaY = r sin ϕ whre ϕ (phi) is the angle between the x-axis and the far end of the arc, and the radius (r) is measured in pixels. Figure 4-20 illustrates these points.

The result should be rounded up to the next bigger integer, since the GDC can draw only an integer number of pixels. This value can be used for the dot count (DC) parameter.

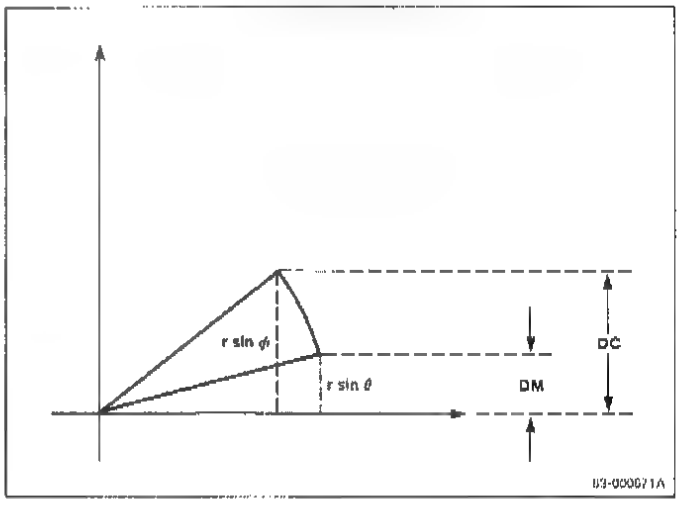
The other arc length that must be calculated (but not drawn) is the number of pixels as the DDA makes its way from the axis to the start of the arc. It is calculated in the same manner as the total dot count, but uses the angle θ (theta) from the axis to the arc. Rounding down the result will ensure that the drawn arc will cover all the pixels it touches. This value is sent to the GDC as the DM parameter.

The other two parameters required for arc drawing describe the radius of curvature (r) to the DDA. The equations for all the parameters are given below:

DC = r sin ϕ (Round up)
D = r − 1
D2 = 2 (r − 1)
D1 = −1 (In two's complement notations = all ones)
DM = r sin θ (Round down)

Radius r is expressed as an integer number of pixels. Note that the D1 parameter value is the same as its initial value. Since no parameters can be skipped during the loading process and DM must be set, the initial value of D1 must be included in the sequence.

Figure 4-20. Determining Arc Length



For circle drawing, ϕ will equal 45-degrees and θ will equal 0 degrees. Therefore, DC = r sin 45° = r (0.707), and DM = 0. It is interesting to note that when drawing circles or adjoining arcs, the common starting point on the axis will be written twice, once for each adjacent arc segment. If complement mode is used for the RMW operation the first arc will set this common bit and the second arc will clear it back to the way it was before drawing. In this situation, using dots masked (DM) of 1 for the second arc will prevent this pixel from being drawn twice.

The four possible starting points for arc drawing, expressed as (x,y) ordered pairs, are:

(Radius, 0)
(0, Radius)
(−Radius, 0)
(0, −Radius)

The above assumes that 0 is the center of the circle or curvature. To move away from the origin, it is necesary to add the coordinates of the offset center to the above coordinates:

(Radius + CenterX, CenterY)
(CenterX, Radius + CenterY)
(CenterX − Radius, CenterY)
(CenterX, CenterY − Radius)

where CenterX is the x-coordinate of the center, and CenterY is the y-coordinate of the center.

RECTANGLE DRAWING

The GDC can draw four line segments in the form of a rectangle with a single CURS, FIGS, and FIGD command sequence. (This assumes that the drawing pattern and RMW operation have previously been set.) Starting from the cursor position, the DDA draws four line segments in series with a counterclockwise 90-degree turn at each intersection. The initially specified direction may be any one of the eight DIR values, so that the figure drawing may start in any corner of rectangles which are either horizontal or 45-degree rotated.

Rectangle drawing requires five parameters with the FIGS command. The only calculation to be made by the host microprocessor is the number of pixels to be drawn on each side of the rectangle. This is passed to the GDC in the form of two values. The first value specifies the number of pixels in the direction given by the DIR parameter, which will be along the first side of the rectangle to be drawn. The second value specifies the length of the second side to be drawn, which will be at right angles to the first side. These two values can be thought of as the height and width of the rectangle. The exact values to be sent to the GDC as parameters to the FIGS command are:

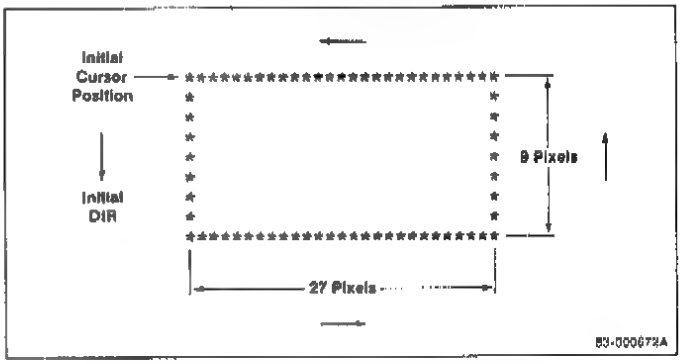
DC = 3
D = (Number of pixels in the initially specified direction) − 1
D2 = (Number of pixels in the perpendicular direction) − 1
D1 = −1
DM = Same value as parameter D.

A sample rectangle would be drawn as in figure 4-21.

Asterisks indicate pixels modified during drawing, and the arrows show the successive drawing directions used during the drawing process for the parameters given below:

DIR = 0
DC = 3
D = 8
D2 = 26
D1 = −1
DM = 8

Figure 4-21. Example of Rectangle Drawing



PATTERNS AND CHARACTERS

The contents of parameter RAM bytes 8 through 15 can be transferred into display memory starting at any pixel position. When filled with the desired pattern or character, the parameter RAM acts as a temporary holding buffer which, until overwritten, can be moved into display memory as many times as desired, to any pixel position, and in any of the eight 45-degree rotation orientations selected by the DIR value. In addition, the pattern may be magnified at the time of writing by a zoom factor from 1 to 16. The enlargement is accomplished via pixel replication in both the X and Y directions.

This drawing capability fills all the pixels in a rectangular or slanted rectangular area based on the parameter RAM contents. The RAM is scanned from the LSB of byte 15, toward its MSB, then up to the MSB of byte 14, toward its LSB, up to the LSB of byte 13, etc, until the MSB of byte 8 is output. If the area to be filled is larger than 8-by-8 pixels, this scanning process is continued until the area is filled. When the area is not a multiple of 8-by-8 pixels (or if smaller than 8-by-8), a subset of the Parameter RAM is scanned out to the bit-mapped display memory, so that the orderly pattern is preserved as far as is possible in the writing area.

The pattern may be drawn into the bit map zoom-magnified up to 16 times. Every integer between 1 and 16 can be used as the magnification factor. The enlarging is done through pixel replication, so that with a factor of 2, each bit in the Parameter RAM is drawn as four pixels of the same value arranged in a 2-by-2 block. Once the pattern is written into the display memory at a certain zoom writing factor, further changes to this zoom factor will not change the previously written patterns.

This facility is controlled by the zoom command. The zoom command also controls the zoom magnification factor for the raster-scan display process. The first (and only) parameter to the zoom command uses the upper four bits for the display zoom factor and the lower four bits for the writing zoom factor.

When the display is zoomed (as differentiated from the drawing zoom factor), everything on the screen is magnified. This is different from the graphics character writing zoom, which only affects the graphics character while it is being written. In contrast, the display zoom factor may be set to magnify the entire display without altering the contents of the bit-mapped display memory. This display zoom factor may then be changed at any time to any other value and the entire screen will immediately reflect the difference. The graphics character zoomed writing feature controls the actual size of the character in the bit-map, and once the character is drawn, it will always stay the same in the bit-map, regardless of further changes in this writing zoom factor.

The parameters to the FIGS command for area filling are:

DC = (Number of pixels in the perpendicular direction) − 1
D = Number of pixels in the initial direction
D2 = Same as parameter D

The rest of the parameter values can be left at their initialization values by not outputting any further parameter bytes before the GCHRD command is given.

Slanted graphics characters and area filling can be used to produce an italics-like effect. This mode of drawing is controlled by a bit in the first parameter byte of the FIGS command. The effect can best be seen in an example (figure 4-22) where the 8-by-8 pattern for the letter "E" is first loaded into the Parameter RAM and then written with no magnification, first in regular fashion and then slanted. An asterisk represents an individual modified pixel. When written with a zoom magnification factor of 2, the same PRAM character pattern will appear as in figure 4-23.

Figure 4-22. Regular vs Slanted Graphics Characters

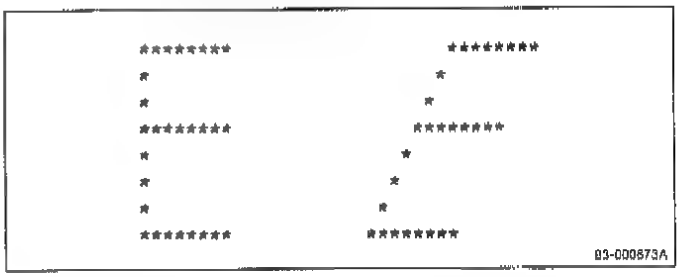
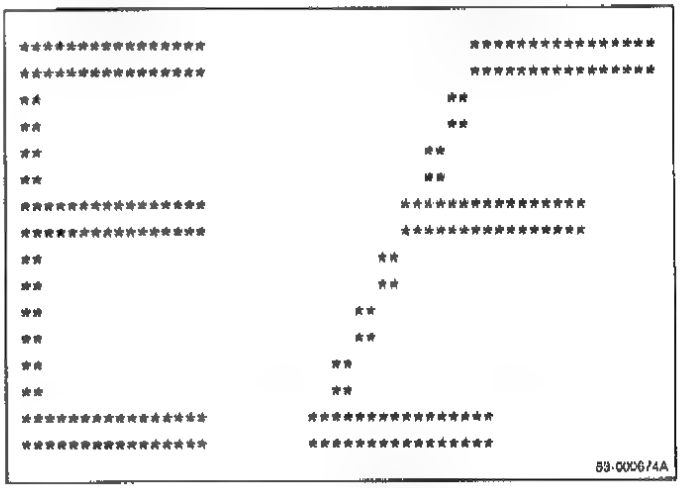


Figure 4-23. Zoom 2 Graphics Characters



When the size of the area to be filled is smaller than the 8-by-8 parameter RAM buffer, the appropriate area in the parameter RAM is drawn into the display memory. Starting from the LSB of PRAM byte 15, as many bits are drawn into the display memory (along the DIR direction) as the D and D2 FIGS parameters allow. As many bytes in the parameter RAM will be accessed as the DC FIGS parameter specifies. These additional bytes are also scanned only as far as the D and D2 parameters allow. The effect is to draw into the display memory only the corner of the PRAM buffer which includes the LSB of byte 15.

In the example of figure 4-24, DIR = 6, so that byte 15 is drawn at the top of the character and bit 0 is placed in the right top corner. This operation will take 35 RMW cycles, instead of the 64 cycles needed to draw the entire PRAM buffer into display memory.

When the size of the area to be filled is larger than the parameter RAM buffer, the buffer is repeated in an X-Y extension fashion to fill the space. Figure 4-25 is an example of an asymmetrical pattern to show the pattern's orientation and representation when it is drawn into the bit-map.

DMA TRANSFERS

DMA transfers into and out of display memory are done using almost the same sequence of commands that are used for the figure drawing operations. The difference comes in the parameters that are given with the FIGS command, and the replacement of the drawing starting commands (FIGD or GCHRD) with the DMA read or write commands: DMAR or DMAW. The DMA transfers move a rectangular area in display memory where the height, width, and orientation is specified in the FIGS command's parameters. These transfers work on a word-address basis only, ignoring the dot address, and one byte is transferred every RMW cycle. The DMAR and DMAW command bytes contain a field to specify whether to access both bytes of each display memory word, just the upper byte, or just the lower. For DMA write operations, the type of RMW operation can be selected via the MOD field in the DMAW and DMAR command bytes.

The DMA facility transfers data to and from the display memory without disturbing the contents of the GDC's FIFO.

Preparing for a DMA Transfer

In preparation for a DMA transfer, the cursor must be pointed to the first display memory word address to be accessed. After this is accomplished, it is necessary to set the mask register to all ones before the DMA transfer is begun. The GDC's DDA advances to the next word address horizontally only when the appropriate edge bit of the mask register is a one. By having a one in both the bit 0 and bit 15 positions no matter how the mask register is rotated, the desired advance of the EAD word address will take place, regardless of the drawing direction.

Figure 4-24. Parameter RAM Contents for Drawing a 5-by-7 Character

	Bits							
PRAM Byte	7	6	5	4	3	2	1	0
15	0	0	0	1	1	1	1	1
14	0	0	0	1	0	0	0	0
13	0	0	0	1	0	0	0	0
12	0	0	0	1	1	1	1	0
11	0	0	0	1	0	0	0	0
10	0	0	0	1	0	0	0	0
9	0	0	0	1	1	1	1	1
8	0	0	0	0	0	0	0	0

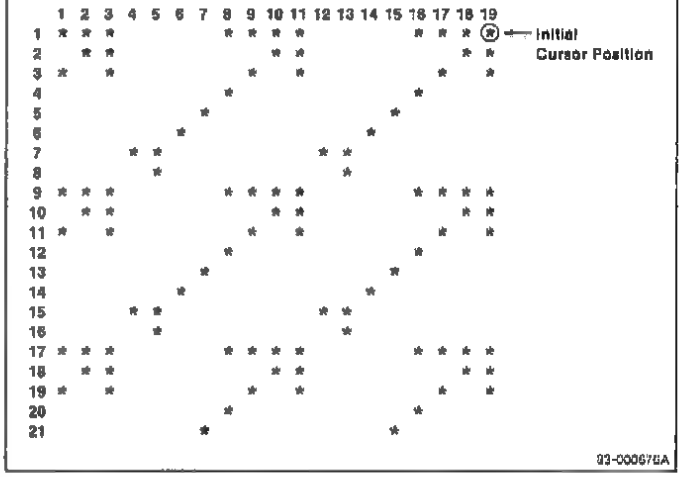
Character as seen on the CRT screen

```
*****
  *
  *
  ****
  *
  *
  *****
```

83-001525A

Figure 4-25. Filled Area Larger than Parameter RAM Buffer

	Bits							
PRAM Byte	7	6	5	4	3	2	1	0
15	0	0	0	0	1	1	1	1
14	0	0	0	0	0	0	1	1
13	0	0	0	0	0	1	0	1
12	0	0	0	0	1	0	0	0
11	0	0	0	1	0	0	0	0
10	0	0	1	0	0	0	0	0
9	1	1	0	0	0	0	0	0
8	0	1	0	0	0	0	0	0



83-000676A

The FIGS parameters for DMA transfers are very straightforward. Similar to the area filling parameters, the DMA parameters require the DIR initial direction and the height and width of the area to be transferred. For odd DIR values, the area transferred will be rotated 45 degrees to the x and y axes. In this case, the address for the next RMW cycle will be calculated by adding to the current EAD word address (pitch + 1) or (pitch − 1) to go down and sideways. To go up and sideways, one of these values is subtracted. For example, the path across the display screen during a DMA transfer in DIR = 3 direction would be:

3rd Word (——)
2nd Word (——)
1st Word (——)

where (——) = a 16-bit word in display memory in the position it would appear in on the CRT screen.

The DC, D, and D2 (only for DMA word reads) parameters control the dimensions of the rectangle of words to be accessed. The DC value selects the number of word addresses to be accessed along the direction at right angles to the initially specified DIR direction. Its value is one less than the number of word addresses and will be the same whether accessing both bytes, only the low byte, or only the high byte.

The D parameter controls the number of bytes to be transferred in the initially specified direction, DIR. If it is desired to move the top bytes of four words, D should be set to 4 − 1 or 3. If it is desired to move both bytes of the same four words, D should be set to (4 x 2) − 1 or 7, since 8 bytes must be transferred in the four words. Of course, if both bytes of each word are to be transferred, D will always be an odd number (2n − 1), while if only one byte is to be transferred from each word, D can be even or odd. These FIGS parameter values are summarized as follows:

DC = (Number of word addresses in the direction at right angles to the initially specified DIR direction) − 1

D = (Number of bytes to be transferred in the initially specified DIR direction) − 1

During word transfers, first the low data byte and then the high data byte is transferred before the GDC goes on to the next word address, regardless of drawing direction.

The DMA word read operation is handled differently. In addition to DC and D, it requires the D2 parameter, and D is calculated slightly differently in that it is two less (rather than one less) than the number of bytes in the initial direction. The D2 parameter is simply one-half of the D value.

DC = (Number of word addresses in the direction at right angles to the initially specified DIR direction) − 1

D = (Number of bytes to be transferred in the initially specified DIR direction) − 2

D2 = D/2

DMA Examples

The examples of DMA transfers in figure 4-26 show words in display memory with a vertical line separating the low byte on the left from the high byte on the right. The enclosed numbers identify the bytes transferred, in order of transfer.

If the DC value is 0, the DMA transfers will be a one-dimensional sequence of accesses, rather than in a two-dimensional area. In this case, D controls the number of bytes to be transferred, up to 2^{14} or 16,384 bytes.

Partial Word DMA Writes

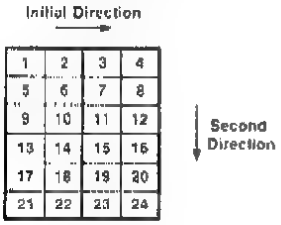
It is possible to transfer data through DMA into the display memory and modify only some of the bits of each word accessed. If only the lower or upper eight bits are to be modified, the appropriate byte transfer mode can be used as described above. If an arbitrary number of bits must be modified, a different approach must be used.

Modifying an arbitrary pattern of bits in each word with DMA word writes can be done one of two ways. With either technique, the mask register must be used to select which bits are to be modified. The trivial case is when only one word is to be modified. The more interesting case is when the vertical boundary of an area to be transferred into does not fall on a word boundary.

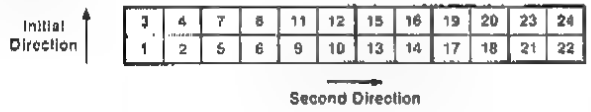
The key element in making partial word DMA writes into display memory is the mask register. In addition to enabling bit modification, the mask register also works with the DDA to control the advancing of the EAD word address. When the instantaneous direction of the next pixel to be drawn has a component along the x axis, the appropriate mask register edge bit is tested for a one to see if the end of the word has been reached. If it has, the EAD is advanced to the next word and the mask register is rotated to point at the first bit to be modified in the new word. If the end of the word has not been reached (the edge bit = zero), the mask register is shifted to point to the next pixel in the current word.

Figure 4-26. DMA Transfers

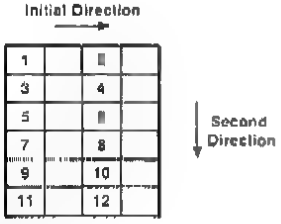
WORD Writing into Display Memory with DIR = 2, DC = 5, and D = 3



WORD Writing into Display Memory with DIR = 4, DC = 5, and D = 3

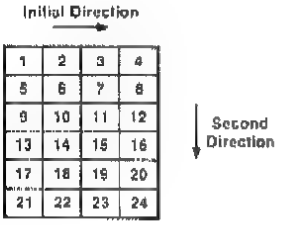


Low Byte Transfer with DIR = 2, DC = 5, and D = 1

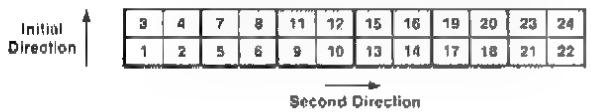


High Byte Transfer with DIR = 4, DC = 5, and D = 1

WORD Reading from Memory with DIR = 2, DC = 5, D = 2, and D2 = 1



WORD Reading from Memory with DIR = 4, DC = 5, D = 2, and D2 = 1



83-0015268

This testing is done for all the drawing directions except straight up and down, in which there is no horizontal component. In these two directions, the EAD will be advanced after every RMW cycle, regardless of the mask register contents and the mask register will not be rotated. To keep the DDA advancing only vertically throughout the duration of the DMA operation, a one-dimensional transfer must be programmed by the FIGS command; the DC value must be 0.

This mode of operation is perfect for DMAing data into the left and right boundary edges of a larger DMA rectangle. The area will have to be filled with three DMA operations: the left edge words, the center rectangle of words, and the right edge words. In this way, an arbitrary rectangle can be DMA-filled on any bit boundary in the display memory.

SUCCESSIVE ADDRESSES

The DDA can also be programmed to increment linearly through successive display memory addresses, in the specified direction. In this mode, the DDA steps through memory one address at a time, for as many accesses as are specified by the DC parameter of the FIGs command. The first parameter byte of the FIGS command sequence should have none of the figure type select bits set for this mode, and only the DC parameter need be sent to the GDC to specify the number of words to be accessed. Any number of RMW cycles can be specified from 1 to $2^{14} = 16{,}384$.

This capability can be used to draw single pixels into the display memory or to access a number of words, for either reading or writing, in either graphics bit-mapped or coded character areas.

When preparing to do an operation involving one RMW cycle per word address, the mask register must be set to all ones with a mask command. This must be done after the cursor is positioned and before the FIGS command. The DDA will then advance the EAD after each RMW cycle because both the LSB and the MSB of the mask register will be one. An exception would be single pixel writing, when the CURS command will load the mask register with one-of-sixteen bits.

WORD READING AND WRITING

Using the successive addresses mode, either or both bytes of each word can be accessed for word reading and writing. Unlike DMA transfers, the data is passed through the command and parameter FIFO under the direct control of the host microprocessor. This is very useful for initializing large amounts of display memory to the same value, such as all zeros or all ones, without the overhead of setting up a DMA transfer. If the DIR value is set equal to 2, the DDA will calculate a path

through ascending addresses for as many words as the DC parameter calls for (DC = number of words − 1). As it crosses the end of a line of memory words, it will go to the next higher address, which is the first memory word of the next line. Each FIGS/WDAT command sequence can access 16,384 words, since the DC parameter has 14 bits. This is the fastest way to clear display memory, because there is no interaction with the host system during 16K words of clearing. It is also a good way to read a range of memory during area flooding.

To use this mode it is necessary to point the cursor to the first word to be accessed and to set the mask register to all ones. This allows the DDA to advance to the next word address after each RMW cycle, regardless of drawing direction.

The DC value specifies the number of words accessed, regardless of the number of bytes accessed per word.

In a graphics bit-mapped area, the WDAT command uses only the least significant bit of the parameter bytes which follow it. In a coded character situation, all the bits of the parameters are used for the RMW operation.

In order to write a random pattern into a word in display memory it is necessary to use the mask register to hold the pattern and pass all-ones to the DDA with a WDAT command. Without an all ones value in the mask register, the EAD will not advance properly to the next word address, except in the straight up and down drawing directions (DIR = 0 or 4), so the cursor must be set for each word.

For a WDAT command, the DC count of the FIGS command is used to control the number of times the first set of parameter data is written into the display memory. This value is decremented in the process of doing these writes until a value of zero is reached. Subsequent sets of parameter data following the WDAT opcode will see DC values of zero. For example, if the FIGS command sets DC to 5 and three sets of data follow the WDAT opcode, the first set will be written into six different words while the second and third sets will be written once each into two additional locations.

GDC COMMAND SEQUENCE

The examples described below are typical of the operations done with the μPD7220 graphics controller in a bit-mapped graphics application. Many other operations are possible and reasonable. Many other modes of operation are possible also. The ones discussed here are intended to be informative and to illustrate basic principles that can be applied to any situation.

Assumptions: 5-MHz GDC clock
800 ns per byte to load the FIFO

Minimum FIFO Load Times

The minimum FIFO load times give an idea of the time required to get the command opcodes and their parameters into the GDC's FIFO. These numbers give an idea of how much time the loading process might take when compared to the parameter preparation time, and the execution time.

These considerations are interesting when considering balancing the various components of your graphics system. For example, it wouldn't make much sense to spend a large percentage of your system's cost on an advanced processor which could calculate vector drawing parameters in 2 μs when the GDC can only accept them over 12 μs. (Of course, if other jobs could be found for the processor to do during the remainder of the time, there might be a motivation to do this.)

The minimum load times have been calculated simply by multiplying the number of bytes to be transferred by 800 ns per byte transferred. This assumes that the FIFO starts empty (and doesn't overflow) and that every other timing factor is optimal. Few if any real systems will achieve sustained transfer rates at this speed.

Initialization [Master VSYNC Mode, Graphics Mode]

This command sequence need be done only after power up. The basic operation of the GDS is characterized to the individual application with these commands and parameters. Other modes of operation will require different parameter bytes and considerations, but the same commands.

- 23 bytes to load into the FIFO
- Minimum FIFO load time of 18.4 μs

RESET Opcode (00H)
P1 = mode bits
P2 = active words per line − 2
P3 = VSYNC and HSYNC − 1 widths
P4 = HFP − 1 and VSYNC widths
P5 = HBP − 1 width
P6 = VFP width
P7 = active lines per video field
P8 = VBP width and top bits of active line count

RESET puts the GDC into the idle mode and prepares it to accept commands and parameters. When determining these parameters, it is important to observe the constraints listed later under the "Initializing Command Sequence" heading.

VSYNC opcode + master/slave bit

VSYNC specifies master or slave video operation.

PITCH opcode
P1 = display memory width in words

PITCH establishes the width of the display memory. For proper display, the minimum pitch should be equal to the display width. Note that while the AW parameter of the RESET command requires a value 2 less than the actual display width, the PITCH command does not subtract 2 and must be loaded with a value greater than or equal to the AW parameter plus 2.

PRAM opcode + PRAM starting address (0 here)
P1 = display window 1 starting word address, low byte
P2 = display window 1 starting word address, high byte
P3 = window 1 length low bits + starting address top bits
P4 = mode bits + window 1 length top bits

PRAM loads the parameter RAM with the desired display starting addresses and lengths. The sum of the partition lengths should be greater than or equal to the number of lines on the display. When in graphics mode, there are two usable display areas. If the sum of the lengths of these areas is less than the number of display lines, the data stored in parameter RAM bytes 8 through 11 will be interpreted as another display partition when in actuality, these bytes are the drawing pattern register and the first two bytes of the graphics character RAM.

When operating in character mode and the sum of the four usable partitions is less than the display length, after the GDC has used up these partitions, it will "recycle" this data and display the first display area as a fifth partition, and so on until the display line count has been reached.

Use of this command when the GDC is programmed for interlaced mixed mode should be limited to display of multiple graphics areas or multiple character areas only. Graphics and character areas may not be mixed in this mode.

CCHAR opcode
P1 = sweep lines per character row −1
= 0 (graphics mode)
P2 = Cursor blink rate lower bits, steady cursor bit, and cursor top line number.
= C0H (interlaced graphics mode)

CCHAR sets up the lines-per-row and cursor blink rate values. It should be noted that if the GDC is programmed to operate in an interlaced display mode, the cursor blink rate must be set to three (3) in order to work correctly.

In the character mode or character area of mixed mode, if a certain combination of cursor parameters is selected, the Ctop parameter is assumed to be zero, causing difficulty with the cursor positioned in the first character line (the top of the screen.) To avoid the problem the following limitations should be maintained.

AL = N x LR
where:
AL = parameter of the RESET/SYNC command
N = number of character lines per frame
LR = number of lines per character
Cbot = LR −1

Zoom opcode
P1 = display + writing zoom magnification factors

ZOOM sets the display zoom factor. Even if no zoom hardware has been implemented, this command must be used to initialize the GDC's internal zoom circuitry.

START opcode

The START command causes the GDC to begin outputting syncing information and the beginning of active display.

Display Mode Change

This command is used to change the mode of operation without resetting the video sync cenerator or flushing the contents of the FIFO. Typically it is used to enter and exit flash mode from flashless mode as drawing requirements change during interactive editing sessions, and to turn on and off the automatic dynamic RAM refresh to maximize the time available for drawing if the RAMs can be otherwise refreshed. The SYNC opcode can also be followed by the complete set of parameters accepted by the RESET command.

- 2 bytes to load into the FIFO
- Minimum FIFO load time of 1.6 μs

SYNC opcode + DE bit
P1 = mode bits

Blank the Screen

This command and the next are especially useful during flashless drawing mode to allow screen-clearing operations to proceed at flash mode-like speeds. By blanking the screen, the GDC will be able to use the entire video field for RMW operations instead of only the horizontal and vertical retrace blanking intervals. The HSYNC interval will continue to be used for dynamic RAM refresh cycles, if so enabled, regardless of whether the screen is blanked or not. The 16

FIGS/WDAT word-clearing command sequences needed to clear a 256K maximum display memory can be executed in under 250 ms using this technique (closer to one-fifth of a second).

If the host processor system can keep the GDC busy doing figure drawing, the initial drawing of the screen can be done at very high speed in flash drawing mode by blanking the screen during this period. Normal interactive drawing can be done in flashless mode, without screen disturbances and little impact on throughput, due to the low overall throughput requirements of direct interaction with the user.

During flash drawing mode operation it may be desirable to blank the screen for lengthy drawing operations rather than allow the user to see a highly disturbed display.

Additionally, a blanked screen can be used as part of the interactive dialogue between the system and the user.

- 1 byte to load into the FIFO
- Minimum FIFO load time of 0.8 μs

BCTRL opcode + DE bit = 0

Unblank the Screen

This command sequence is the inverse of the screen blanking sequence described above.

- 1 byte to load into the FIFO
- Minimum FIFO load time of 0.8 μs

BCTRL opcode + DE bit = 1

Drawing Setup

Before any drawing can be done, the drawing pattern and the RMW operation must be established. They need not be set again unless a change is desired or a value is disturbed by another command. The drawing pattern is stored in the parameter RAM by the appropriate PRAM command. Note that the locations in the parameter RAM used by the drawing pattern (RA-8 and RA-9) are also used by the area filling (graphics character) patterns (R-8 through R-15). Also note that several GDC command opcodes have the RMW operation selecting field, MOD. The operation may be inadvertently changed during a DMA read operation, for example, when the MOD field is otherwise of no concern.

- 4 bytes to load into the FIFO
- Minimum FIFO load time of 3.2 μs

WDAT opcode + transfer type + RMW operation, code

PRAM opcode + PRAM starting address of 8
P1 = drawing pattern, low byte
P2 = drawing pattern, high byte

Cursor Positioning [Graphics Mode]

All RMW operations, whether figure drawing, DMA transfers, or data reads and writes, use the cursor position as their starting location. Occasionally the cursor value from the previous operation will be correct for the next operation, but normally the cursor will have to be reloaded with a new value for each RMW operation. For example, if done correctly, it is possible to chain vectors end-to-end without updating the cursor between vectors. It is also convenient to do this with area filling and data reading and writing.

The cursor value in graphics mode has two parts: the address of the word in memory (18 bits), and the address of the bit (pixel) within that word. The EAD (exection address) register holds the word address and the mask register holds the dot address, dAD. Notice that the CURS command and the mask command interact in that they both change the contents of the mask register. The CURS command loads the mask register with the one-of-sixteen value which corresponds to the four-bit dAD value of the fourth parameter byte.

- 4 bytes to load into the FIFO
- Minimum FIFO load time of 3.2 μs

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

Mask Register Loading

The mask command loads the mask register with the 16-bit value specified by its parameter bytes. This command should only follow the CURS command since dot address dAD will overwrite any previous contents of the mask register.

- 3 bytes to load into the FIFO
- Minimum FIFO load time of 2.4 μs

MASK opcode
P1 = least significant byte of MASK pattern
P2 = most significant byte of MASK pattern

Vector Drawing

This command sequence positions the cursor to the figure's starting pixel address in display memory, sets up the digital differential analyzer (DDA), and initiates the drawing process. It is not necessary to load the cursor address if the previous operations have left it at the starting point of the desired vector.

- 15 bytes to load into the FIFO
- Minimum FIFO load time of 12.0 μs
- Execute time for 100 pixels = 80 μs

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

FIGS opcode
P1 = type = 00000 (binary) + DIR
P2 = GD bit + low byte
P3 = DC high byte
P4 = D low byte
P5 = D high byte
P6 = D2 low byte
P7 = D2 high byte
P8 = D1 low byte
P9 = D1 high byte

FIGD opcode to start drawing process

Single-Dot Writing

When drawing just a single pixel (or word) it is not necessary to supply all the DDA parameters, since it will be doing only the first RMW operation. The number of bytes sent to the GDC can, therefore, be reduced.

- 7 bytes to load into the FIFO
- Minimum FIFO load time of 5.6 μs

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD bits (0-3) + EAD (bits 16-17)

FIGS opcode
P1 = type = 00000 (binary) + DIR

FIGD opcode to draw the dot

Arc Drawing

The considerations for this command are similar to those of vector drawing described previously.

- 17 bytes to load into the FIFO
- Minimum FIFO load time of 13.6 μs

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

FIGS opcode
P1 = type = 00100 (binary) + DIR
P2 = GD bit + DC low byte
P3 = DC high byte
P4 = D low byte
P5 = D high byte
P6 = D2 low byte
P7 = D2 high byte
P8 = D1 low byte
P9 = D1 high byte
P10 = DM low byte
P11 = DM high byte

FIGD opcode to start the drawing process

Random-Size Areas and Characters

This command sequence is useful for filling random-size areas on the screen and for writing into the bit-map characters which are other than 8 by 8 bits. A random area of up to 2^{14} (16,384) pixels along each edge can be filled with one command sequence. The area will be filled mozaic-style from the pattern loaded into the parameter RAM by replicating the pattern in both the x and y dimensions. When drawing bit-mapped characters which exceed 8 bits in either direction, it will be necessary to reload the parameter RAM for the bits outside the basic 8 by 8 square.

When drawing 7 by 9 characters in a 9 by 11 (or larger) window, it is not necessary to reload the parameter RAM to draw the extra column of bits beyond the first eight bits of the character window's width. The ninth bit of the character window's width can be written as a repeat of the first bit in the row. The first bit of each row will always be zero since they are the "guard" bits of the window around the actual character. In this case, the GDC would be programmed to draw a nine-bit wide area and it will use this first zero bit in both the first and ninth positions. The rows of characters after the eighth will, of course, require the reloading of a part of the Parameter RAM.

Setup for this command requires the establishment of the RMW operation, and the writing zoom factor.

Be sure that the last pattern byte to be loaded with the PRAM command goes into location 15 of the parameter RAM, since this is where the GDC will get its first byte of information for the filling process.

If the drawing direction and orientation of the pattern bits are coordinated, a succession of drawing operations can be made without reestablishing the cursor position.

It should be noted that when executing RMW cycles for graphic character drawing after every horizontal line of pixels has been written into video memory, the GDC takes six extra clock cycles to calculate the address of the first pixel of the next line. During these six clock cycles, the drawing in progress bit (bit 3) in the status register will go low and then return high for the next line. So, for example, if it is an 8 x 8 graphic character or area fill, bit 3 in the status register will go low seven times.

This may create problems if the software attempts to change the plane address for each drawing command, based on the drawing-in-progress flag. There are two ways to avoid such a situation. One is to insert a deglitching delay at least six clock cycles long in the software. The other way is to follow the GCHRD command with some no operation commands and then check for FIFO EMPTY instead of the drawing-in-progress flag.

Relative order of the CURS and PRAM commands is unimportant and is often dictated by the application's requirements.

- 13 bytes + extra PRAM bytes to load into the FIFO
- Minimum FIFO load time of 10.4 μs + 0.8 μs for each extra PRAM byte after the first

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

PRAM opcode + PRAM starting address
P1 through Pn = pattern bytes

FIGS opcode
P1 = type = 00010 (binary) + DIR (type = 10010 for slanted)
P2 = DC low byte
P3 = GD bit + DC high byte
P4 = D low byte
P5 = D high byte

GCHRD opcode to start area filling process

Area Filling/Graphics Character [8 by 8]

Refer to the comments for random size area filling described above. Similar considerations apply to this command sequence.

- 17 bytes to load into the FIFO
- Minimum FIFO load time of 13.6 μs
- Execute time ≥ 64 x 0.8 = 51.2 μs

PRAM opcode + PRAM starting address of 8
P1 = pattern byte 8 (last drawn)
P2 = pattern byte 9
P3 = pattern byte 10
P4 = pattern byte 11
P5 = pattern byte 12
P6 = pattern byte 13
P7 = pattern byte 14
P8 = pattern byte 15 (drawing starts with bit 0)

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

FIGS opcode
P1 = type = 00010 (binary) + DIR (type = 10010 for slanted)
P2 = GD bit + DC low byte

GCHRD opcode to start area filling process

Rectangle Drawing

The considerations for this operation are similar to those for vector drawing described previously.

- 17 bytes to load into the FIFO
- Minimum FIFO load time of 13.6 μs

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

FIGS opcode
P1 = type = 01000 (binary) + DIR
P2 = DC low byte
P3 = GD bit + DC high byte
P4 = D low byte
P5 = D high byte
P6 = D2 low byte
P7 = D2 high byte
P8 = D1 low byte
P9 = D1 high byte
P10 = DM low byte
P11 = DM high byte

FIGD opcode to start the drawing process

Pan and Scroll the Display Window

Changing a display partition window starting address will cause the window to roam around in the larger display memory. The display memory may be larger than the display window in either or both the x and y directions. It is a good idea to change the window starting address after VSYNC has ended and before active video begins, so that the GDC's command processor will not access a half-new, half-old value when it goes to the parameter RAM for the address for the display scanning process. The vertical back porch is an ideal time to do this modification, because the first partition's parameters have already been accessed but there is no chance of the second partition's values being needed until later. The vertical back porch time can be found by testing the VSYNC status register for a falling edge.

- 3 bytes to load into the FIFO
- Minimum FIFO load time of 2.4 μs

PRAM opcode + PRAM starting address of 0, 4, 8, or 12
P1 = display partition starting address, low byte
P2 = display partition starting address, high byte

Zoom Factor Change

Since the display zoom factor is passed into the GDC in a single byte, the same timing precautions need not be heeded for changing the display partition starting addresses. The writing zoom factor can be changed at any time. The external zoom pixel prescaler counter must also be set to correspond to the zoom factor given to the GDC.

- 2 bytes to load into the FIFO
- Minimum FIFO load time of 1.6 μs

ZOOM opcode
P1 = display factor + writing factor

Set Background

This command sequence is used to initialize large areas of display memory. Simple incremental advancing is done to calculate the next word address in display memory. Note that the setting of the mask register to all ones forces the DDA always to increment the EAD word address after each RMW cycle, and the RMW logic unit to modify all the bits of each word. It is typical to use the set, clear, replace RMW operations when clearing or setting memory. Also note that in graphics mode, this sequence can be used only to set or clear all the bits of each word accessed to the same value (bit 0 of the WDAT parameter).

If more than 16,384 successive words are to be accessed, there is no need to reestablish the cursor and mask values for each FIGS/WDAT command pair.

- 14 bytes to load into the FIFO to fill up to 16,384 words
- Minimum FIFO load time of 11.2 μs
- Execute time $\geq$ 13.107 μs for ■ 16,384 word block
- 7 bytes to fill additional successive areas, each up to 16,384 words long

CURS Opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

MASK opcode
P1 = FF (hex)
P2 = FF (hex)

Repeat

FIGS opcode
P1 = TYPE = 00000 (binary) + DIR
P2 = DC low byte
P3 = GD bit + DC high byte

WDAT opcode + transfer type + RMW operation
P1 = pattern data, low byte (only bit 0 used)
P2 = pattern data, high byte

Until

Multipixel Write

In graphics mode by using the mask register to hold the pattern data, any pattern of bits may be written into a display memory word in one RMW cycle. A dummy pattern is passed as parameters of the WDAT command and must have a one in bit 0 of the low byte to allow the desired pattern in the mask register to express itself in every bit position. Since the mask register also controls whether the DDA increments the EAD address, an arbitrary mask pattern will not always be executed once, since no relevant parameter is changed during this mode of operation. This command will not perform replace operation because the bit pattern in the mask register is used only as masking bits, not as new data. To perform replace function the word would have to be first cleared in the video memory.

- 12 bytes to load into the FIFO for the first word
- Minimum FIFO load time of 9.6 μs for the first word
- 10 bytes to load into the FIFO of each succeeding word
- Minimum load time of 8.0 μs for each succeeding word

FIGS opcode
P1 = type = 00000 (binary) + DIR

Repeat

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

MASK opcode
P1 = pattern, low byte
P2 = pattern, high byte

WDAT opcode + transfer type + RMW operation
P1 = FF (hex), dummy pattern low byte
P2 = FF (hex), dummy pattern high byte

Until

Reading Data from Display Memory

The contents of display memory can be read to the host processor through the FIFO in a linear sequence of addresses of up to 16,384 locations. Any linear direction may be selected from the eight selected by the DIR parameter of the FIGS command. After the cursor is set with the CURS command, it is necessary to set the mask register to all ones for all DIR directions other than 0 and 4 to ensure the proper incrementation of the EAD word address. The GDC's FIFO will be turned around from write mode to read mode when the RDAT command is executed. Any commands or parameters which followed RDAT command into the FIFO will be lost as the read operation begins.

The GDC will begin to do RMW memory cycles to read the requested data soon after the RDAT opcode is executed. The data is loaded into the FIFO byte by byte as it becomes available. After the first byte is in the FIFO, the GDC's data register is loaded with the top byte on the FIFO and the data ready flag in the status register will be set. The host procesor may then read the data at GDC port address 1. GDC internal hardware then gets the next byte from the FIFO, if available, and into the data register, etc. It is important for the host processor to check the data ready status bit before each read, because the FIFO is implemented as a ring buffer RAM and time is required to access its contents. The peak transfer rate from the FIFO will yield a byte from the FIFO every four 2xWCLK cycles, which is the same rate at which the FIFO is filled via RMW memory cycles. Actual transfer rates will be lower in flashless mode since the RMW cycles cannot be done during much of the video field time.

The GDC will continue to fetch data from the display memory into the FIFO until the DC value decrements to zero. These RMW read cycles will pause if the FIFO is full until the host processor makes more room in the FIFO by reading data.

The host processor may abort the read sequence before the total number of words has been accessed by outputting a command byte to the GDC. The FIFO is then forced into write mode to accept and normally process the command output operation and its following parameters. Note that the command output operation, in this situation, should not check the status register FIFO-FULL bit before outputting the command byte. The FIFO may well be full of read data as the command is about to be output. All read data in the FIFO at the time a command byte is received will be lost when the FIFO turns around.

- 12 bytes to load in to the FIFO to initiate a sequence of words to be read
- Minimum of 9.6 μs to initiate the read operation

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = word address dAD (bits 0-3) + EAD (bits 16-17)

MASK opcode
P1 = FF (hex)
P2 = FF (hex)

FIGS opcode
P1 = type = 00000 (binary) + DIR
P2 = DC low byte
P3 = GD bit + DC high byte

RDAT opcode + transfer type + RMW operation

DMA Data Writing

Display memory data can be written via the GDC's DMA interface once the GDC is set up for the transfer. The external DMA controller must also have been programmed for the operation. Once the cursor is pointed to the starting point of the block to be transferred, the mask register must be filled with all ones to ensure that the EAD word address witll be properly advanced as the DMA transfer progresses.

The FIGS command is used to set the TYPE, DIR, DC, and D values. After the DMAW opcode is executed by the GDC's command processor, the DREQ pin will signal the start of the transfer. Any commands which followed the DMAW opcode into the FIFO will not be lost during the DMA transfer since the DMA data bypasses the FIFO. The status register may be read while DREQ is asserted, as long as DACK is not asserted. No commands should be loaded into the GDC while the DMA execute status tegister bit is high.

- 14 bytes to load into the FIFO to Initiate the transfer
- Minimum fo 11.2 μs to set up for the DMA transfer

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

MASK opcode
P1 = FF (hex)
P2 = FF (hex)

FIGS opcode
P1 = type = 00000 (binary) + DIR
P2 = DC low byte
P3 = GD bit + DC high byte
P4 = D low byte
P5 = D high byte

DMAW opcode + transfer type + RMW operation

DMA Data Reading

The preceding comments made for DMA data writing also apply for DMA data reading.

CURS opcode
P1 = word address EAD (bits 0-7)
P2 = word address EAD (bits 8-15)
P3 = dot address dAD (bits 0-3) + EAD (bits 16-17)

MASK opcode
P1 = FF (hex)
P2 = FF (hex)

FIGS opcode
P1 = type = 00000 (binary) + DIR
P2 = DC low byte
P3 = GD bit + DC high byte
P4 = D low byte
P5 = D high byte

DMAR opcode + transfer type + RMW operation

INITIALIZING THE GDC

After power-up, the GDC must be initialized by a series of commands and parameters to configure it for the desired mode of operation. The host microprocessor will output this list of commands and their parameters as part of its power-up routine. Other than checking the GDC FIFO's status to prevent FIFO overflow, only the transfer of fixed values into the GDC is necessary. Once the GDC or GDCs are initialized, the START command starts the display process.

Command Sequence

Several commands must be given to the GDC to configure it for the desired operating mode. They are introduced here and are fully explained in the data sheet.

(1) RESET puts the GDC into the idle mode and prepares it to accept commands and parameters.
(2) SYNC configures the video sync generator and operating modes. (These parameters may follow the RESET command without the SYNC command byte.)
(3) VSYNC specifies master or slave video operation.
(4) CCHAR sets up the lines-per-row and cursor Blink Rate values.
(5) PITCH establishes the width of the display memory.
(6) PRAM loads the parameter RAM with the desired display starting addresses and lengths.
(7) ZOOM sets the display zoom factor.
(8) CURS positions the cursor, which is especially important during initialization of character display modes.

Other than the RESET command which must come first, the order in which these commands are given is not important since they do not interact with each other. The GDC will be in the idle mode until a START command is issued. During idle mode, the display will be blanked and the video timing of any slave mode GDCs will synchronize to their master GDC or external sync source. During this synchronization process, the master GDC's video sync generator will produce and output both horizontal and vertical sync signals, while the slave GDC or GDCs will use their VSYNC pin as an input for this externally generated vertical sync signal. At the end of each VSYNC pulse, the slaves will reset their sync generators to line up with the external signal from the master sync source.

During idle mode, all GDCs will generate non-interlaced video whether or not interlaced video was specified. Once idle mode ends via a START command, all the GDCs will be generating the same video field of the video frame sequence. Once initialized as above, the GDC can accept and execute any of its other commands. The idle mode is ended by a START command, which stops the video sync generator from synchronizing to the external vertical sync signal, and unblanks the display raster.

Once these commands have initialized the GDC, but before the START command, you may wish to clear the display memory. If the GDC is being operated in slave mode, it will need time to synchronize to the master sync source before the display START command is given. The host microprocessor should watch the VSYNC status bit of the master GDC to ensure that at least one complete VSYNC pulse reaches the slave GDC.

After receiving the RESET command, the GDC will start HFP and VBP if the GDC was previously set to be master. If the GDC is a slave, after RESET the GDC will load its HFP with the value 3, six clocks after the trailing edge of the external VSYNC pulse. This process will take place continuously until the START command is issued. From then on, the slave GDC does not sense the VSYNC input anymore.

When calculating video parameters, it is important to keep the following constraints in mind:

Parameter	Options Used	Minimum Value
HFP	Light pen	6
HFP	Slave mode	4
HFP	DMA operations	3
HFP	Zoom and interlace	3
HFP	Default	2
HS	Interlace	5
HS	DRAM refresh	2
HBP	Mixed mode	5
HBP	WD bit set	5
HBP	Interlace	5
HBP	Multiple partitions	5
HBP	Default	3

RESET Command Considerations

The first command to send to the GDC is the RESET command. It is interpreted by special hardware ahead of the FIFO to ensure that, whatever the power-up status of the command processor, RESET will prepare it to accept the command and parameter protocol. The reset operation will also initialize and empty the FIFO, and initialize various internal counters.

The RESET command byte may be followed by the parameters that set up the video sync generator and the basic operating mode. Alternatively, after the RESET command has been given, the SYNC command byte can precede these parameters (without initializing the GDC again).

It is not a good idea to check the FIFO-FULL status bit before outputting the RESET command, since the FIFO may have powered up with the FIFO-FULL bit set. With the command processor waiting for its first command from the FIFO it assumes is empty, the FIFO will never become empty from the point of view of the host microprocessor. Of course, after the RESET command, the FIFO-FULL bit should be checked before each byte is loaded into the FIFO. (It is possible to imagine a GDC driver program sophisticated enough to check the FIFO-EMPTY bit and do its own byte transfer counting, but the speed gained would not offset the increased complexity of such a routine.)

FIFO BUFFER

The main pathway for information flow between the host microprocessor and the GDC is the first-in/first-out (FIFO) buffer internal to the GDC. Commands and parameters are loaded into the buffer by the host and removed at the other end by the GDC's command processor. With this technique, the host can load the commands and parameter bytes as they become available. The GDC command processor then handles them when it finishes execution of the previous command. In this way, the interface between these two asynchronous system elements is made more efficient. Without this low-overhead technique, the interface could be a major performance bottleneck. In addition, the FIFO buffers data for the host as it is read from the display memory or internal registers.

As with all FIFOs, the length of the GDC's FIFO is limited, and if data is output when the FIFO is full, the oldest data in the FIFO will be overwritten and lost. When the host is reading from the FIFO, the data is moved from the FIFO into a temporary data register to allow fast access times onto the system data bus. The possibility of a problem transferring data can be eliminated by checking the status bits that report on the condition of the FIFO and data register. An additional overriding concern is the maximum speed at which the FIFO can transfer data. A byte of data can be transferred no more often than once every four 2xWCLK cycles, even if the FIFO is not full and the data register is ready.

FIFO Status Bits

Three bits relating to the FIFO can be read in the GDC's status register: FIFO-EMPTY, FIFO-FULL, and data-ready. The names of these bits describe their one state condition. For example, the FIFO-FULL bit is zero when the FIFO is not full. The two FIFO status bits are meaningful whether the data is flowing from the host into the GDC or the reverse. The data-ready bit is used only for data reads out of the GDC. None of these bits are meaningful before the first RESET command opcode is sent to the GDC after power-up.

FIFO Operation Modes

When commands and parameter bytes are being written into the GDC, the FIFO is in data write mode. After one of the commands which requests data from the GDC is executed, the FIFO is turned around into data read mode. Bytes of data are then read from the data register, which is in turn filled from the FIFO. The host processor must check the data-ready status bit before each read operation. The FIFO will automatically resume data write mode operation, whether all the data is read or not, when a command byte is output to the GDC. Turnaround of the FIFO to either mode will completely empty the FIFO of any contents.

Command and Data Transfer

The normal sequence of events involves a large number of command and parameter writes (being careful that the FIFO does not overflow), and occasional data reads (being sure that there is data available). The light pen address, the cursor position, and data from the display memory can all be read.

During the command and parameter outputing phase, the host processor works to keep the FIFO as full as possible without causing a data overflow. When data must be read from the GDC, the host processor puts the appropriate command sequence into the FIFO like any other command. It then pauses and waits for the data-ready status bit to go high. While it is waiting, the command processor empties the FIFO of all commands and parameters that preceded the data read command.

The read command opcode byte is then fetched from the FIFO and decoded. Next, the FIFO is turned around and data read mode is entered. There is no lost data during the turnaround operation because nothing was loaded into the FIFO after the read data command opcode.

The GDC then proceeds to fetch the requested data for loading into the FIFO. If the data is coming from the display memory and the flashless mode has been selected, there can be delays during the active display time until the GDC can get the first data byte into the FIFO. The FIFO is then filled as fast as the requested data can be accessed. If the data is coming from the on-chip registers (the cursor or the light pen) the FIFO will be loaded without delay.

After the first byte is put into the FIFO, the process of sending the data to the host processor begins. The data is moved from the host side of the FIFO one byte at a time into the data register. For each byte the data-ready status bit is set to one. The host processor tests this bit until the data is ready and then reads the byte. The data-ready bit must be checked before the next byte can be read from the Data register. If there is data waiting in the FIFO to be read, the GDC needs four 2xWCLK cycles to load each byte into the data register.

The number of bytes to be transferred is determined either implicitly (for the cursor or light pen reads) or explicitly (by the FIGS command for memory reads) before the transfer begins. The host processor has the

option to abort the transfer sequence at any point by simply outputting any command opcode byte to the GDC.

Testing the Status Bits

After power up, the status bits are not meaningful until the RESET opcode is output to the GDC. The RESET will empty the FIFO, initialize the status bits, and prepare the FIFO to receive further data. The RESET opcode is decoded by dedicated hardware ahead of the FIFO, instead of by the command processor after passing through the FIFO. It is important not to check the FIFO-FULL bit before outputting the RESET opcode. The problem is that the FIFO control logic may have powered up with FIFO-FULL set, indicating a full FIFO, so that the status bit will never go low until the RESET command is given.

During outputs to the GDC, the FIFO must not be allowed to overflow. There are two approaches for preventing this. The first is to check the FIFO-FULL status bit for a zero before outputting each command and parameter byte. The second technique is to wait for the FIFO to become empty and then send 16 bytes or less, in sequence, to the GDC. Of course, the maximum transfer rate of one byte every four 2xWCLK cycles must not be exceeded. It is not a good idea to assume the GDC will be able to take more than 16 bytes, hoping the command processor will remove a few bytes while the others are being loaded. The problem comes up when the GDC is busy doing a figure drawing, for example, and takes nothing from the FIFO for long periods of time while it draws.

During data reads from the GDC, the data-ready status bit must be checked before each read operation. The internal structure of the GDC requires a separate data read register to meet the access time requirements of the host interface bus. This is required because the FIFO is implemented as a ring buffer and is dual-ported between the host interface and the command processor. The host interface always has priority for the next FIFO RAM access cycle, but a command processor access is allowed to finish before the next byte is fetched into the data register for the host interface. The data register eliminates any latency delay problems which might arise while accessing the dual-ported RAM FIFO.

A second consideration for data read operations involves early terminations of read sequences. The FIFO-FULL bit may be a one, to indicate a FIFO full of read data, when the host tries to terminate the read sequence. The state of the FIFO-FULL bit must not then keep the command opcode from being issued and turning around and emptying the FIFO. If the remaining data is 16 bytes or more and is filling the FIFO, the FIFO-FULL bit will not go low until either more data is read from the FIFO, or a command opcode is received. In this situation, if the FIFO-FULL bit is checked for a zero before the command is issued, the system will hang. The easy solution is not to test the FIFO-FULL status bit. This, of course, is a potential problem only during reads of display memory data.

DMA INTERFACE HARDWARE

The μPD7220 GDC is designed to interface to an external DMA controller. The DMA interface uses two dedicated pins (DRQ and $\overline{DACK}$), the read and write strobe pins ($\overline{RD}$ and $\overline{WR}$), and the eight-bit data bus pins. Data can either be written from the host system into the display memory or read from the display memory out to the host system memory. A byte of data can be transferred every four 2xWCLK clock cycles during DMA transfer times in the video raster. Note that only video display memory data can be transferred using this interface; GDC commands and parameters cannot be transferred.

The GDC and external DMA controller each supply a memory address in their respective domains for the DMA transfer. The DMA controller typically supplies successive addresses in the host system memory. The GDC, on the other hand, can sequence through a two-dimensional block of addresses in the display memory. The x and y dimensions of this block can each be programmed over a range including the display memory's largest size. The block can also be rotated by any 45-degree multiple. Because of these two separate addresses for each transfer, very flexible DMA transfers can be done.

This DMA capability is very useful for moving data around the display memory via a temporary buffer in the system memory. Multiple windows in the display memory can therefore be easily generated and maintained. It can also be used to read display memory contents into a mass storage device or a printer. In imaging applications, the image can be written into the display memory frame buffer via the DMA facility. Many other applications are possible.

Using external hardware, there are other possibilities. For example, commands and parameters can be DMA'ed into the GDC without the use of the GDC's DMA interface. Another interesting possibility is doing DMA without a DMA controller by substituting the host processor in its role. This is especially interesting when the processor needs a high-speed, low-overhead way to write directly into the display memory.

Basic DMA Cycle

The data to be transferred uses the eight-bit data bus interface but bypasses the FIFO. Commands which follow the DMA command into the FIFO will not be lost since the FIFO is not disturbed. The GDC signals its readiness to do a DMA cycle with its DMA request (DREQ) output. This output going or staying high signifies that at least one DMA cycle can be done. It is high during the times in the video raster when DMA cycles are permitted. Once the transfer with the DMA controller takes place, the GDC does an RMW cycle to write the byte or fetch the next byte.

DMA reads and writes are differentiated from command and parameter writes by the assertion of the GDC's DMA acknowledge input ($\overline{DACK}$) before, during, and after the write ($\overline{WR}$) or read ($\overline{RD}$) strobe assertion. In these operations, the address-0 input of the GDC is ignored. When operating in DMA mode, the DREQ signal will stay high for multiple byte transfers. It will not go high/low/high/low for each data byte request. If the transfer involves a large number of bytes, DREQ will go low for the duration of horizontal blanking and then continue to stay high. If flashless mode is selected, the transfer takes place only during vertical retrace (excluding horizontal blanking). If flash mode is selected, the transfers also take place during active display time. It is important to remember that $\overline{DACK}$ must go low for each $\overline{RD}$ and $\overline{WR}$ pulse and come back to its high level between these pulses.

Once a DMA sequence has been initiated by the host processor, the GDC will assert its DREQ output during the AW interval of lines in which DMA is permitted, until the requested number of transfers has been made. The vertical back porch (VBP) lines will always be used, as will the active lines (AL) if flash mode has been selected.

Because of the restricted time intervals during which DREQ is asserted, only the DMA-execute status bit can be used to determine when the total DMA operation is finished. While a DMA operation is executing, the host should not put more commands into the GDC FIFO. The only interaction with the GDC, beyond the DMA transfers, should be to read the status register.

DMA Cycle Minimum Length

When a partial rectangle area of video memory is selectively specified for the DMA transfer, and the DMA transfer is performed one byte at a time:

1 DMA cycle = 5 x t_{CLK}

In other cases (when the transfer is in 16-bit words):

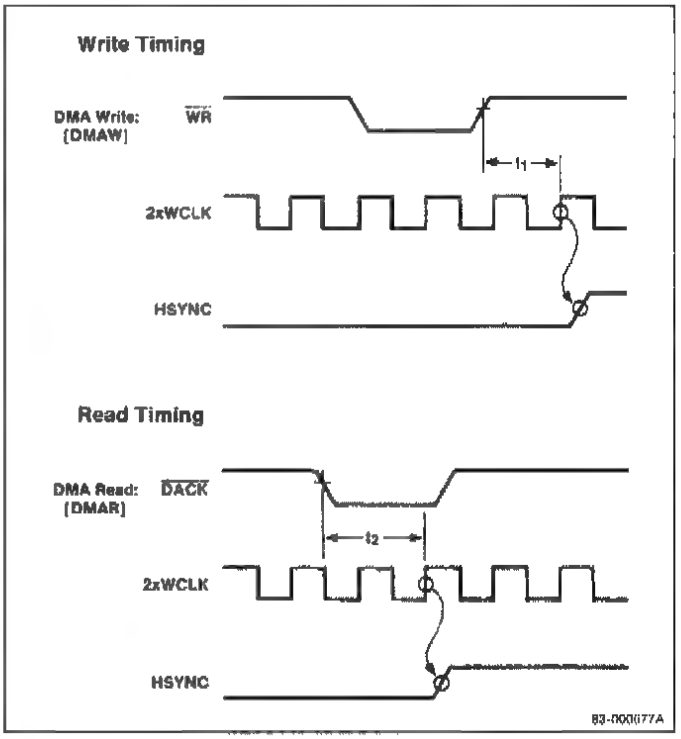
1 DMA cycle = 4 x t_{CLK}

DMA Transfer Timing

The following timing requirement (figure 5-1) should be met in relation to t_1 and t_2. If t_1 is met, t_2 will be met automatically.

$t_1, t_2 \geq t_{CLK}$

Figure 5-1. DMA Timing



From this requirement, the following timing consideration becomes necessary. When the clock frequency of the GDC is the same as that of the DMA controller, and the device that requests DMA transfer is only the GDC:

Horizontal front porch (HFP) $\geq$ 3 characters (or 2 when clock phase matching is good between the GDC and the DMA controller)

In figure 5-2, which illustrates the reason for this requirement:

HFP = 1 character (2 clocks)

GDC clock = DMA clock

In addition to the previous considerations, an external circuit (figure 5-3) may be necessary depending on the system configuration.

Figure 5-2. DMA Timing Waveforms

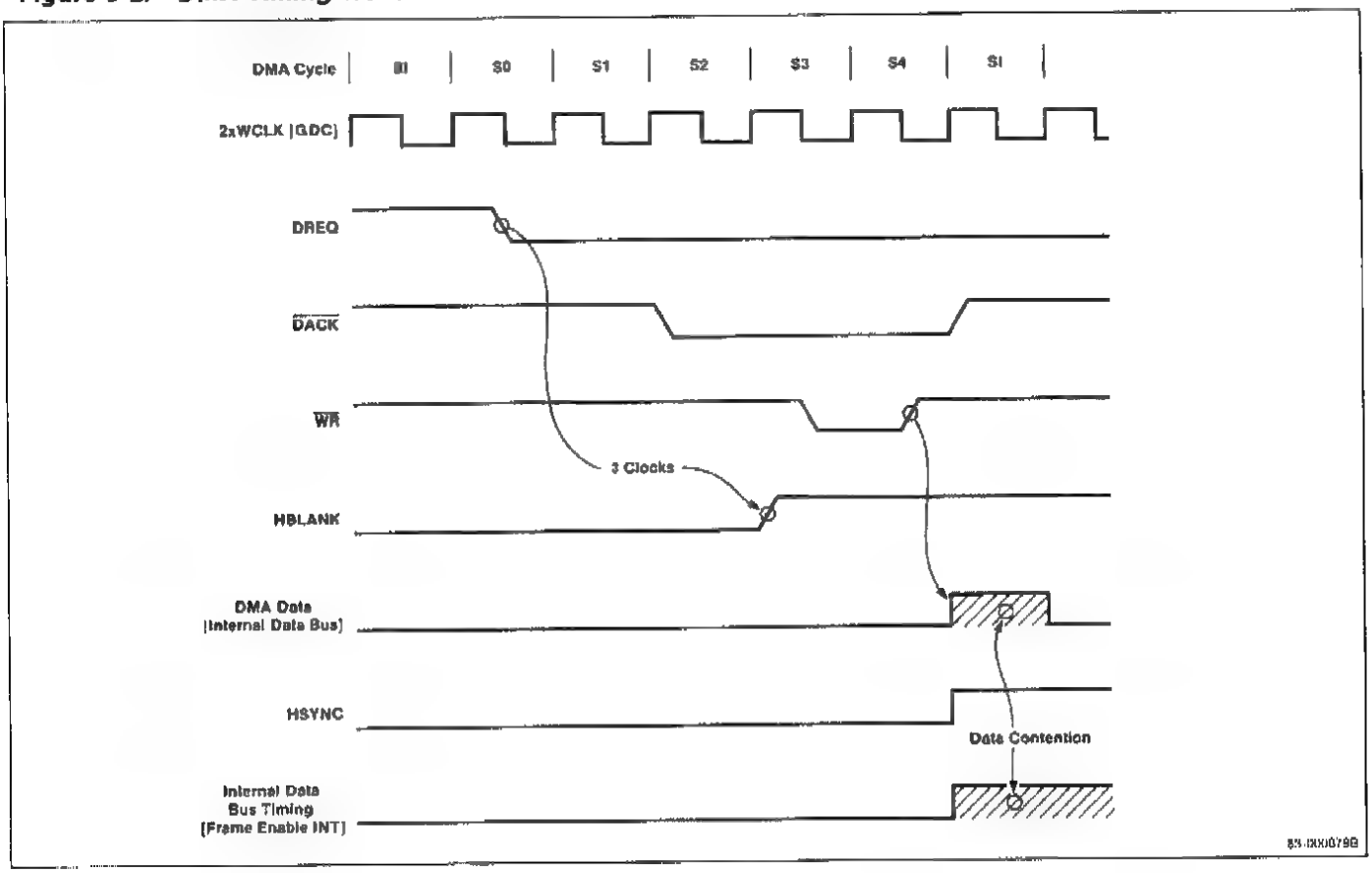
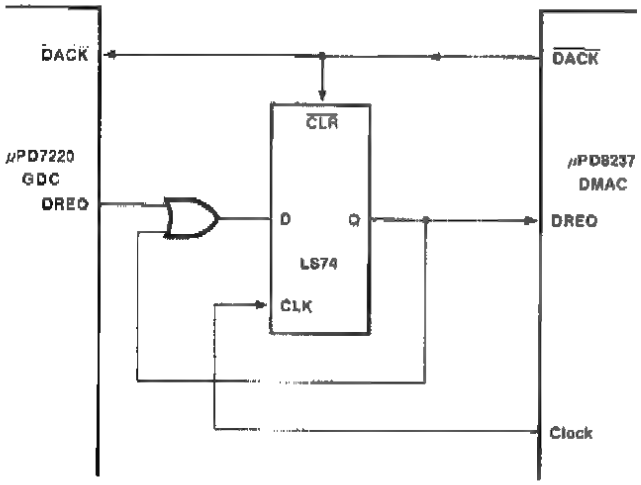
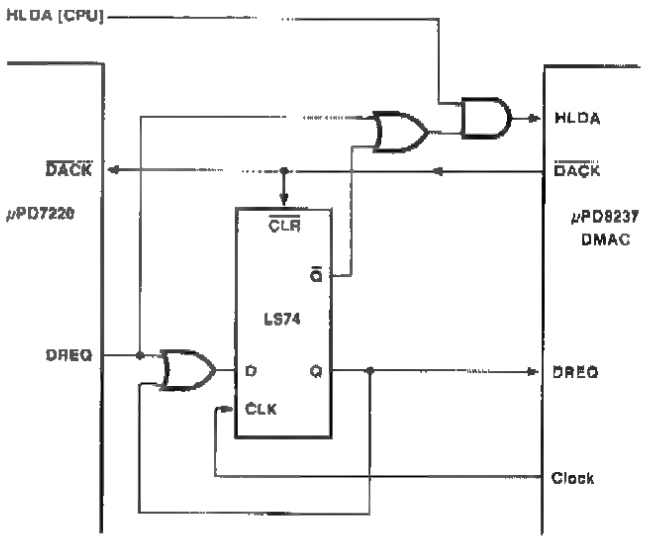


Figure 5-3. Operation with μPD8237 DMA Controller

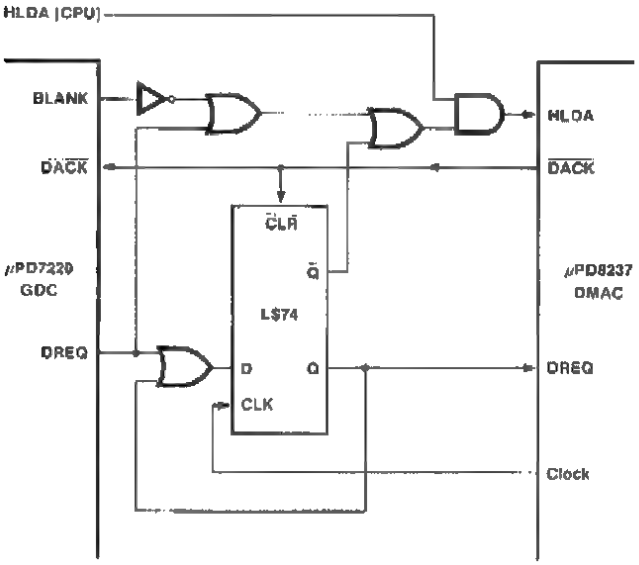
a. Configuration to Maintain DMA High Until $\overline{DACK}$ Occurs



b. Configuration to Maintain $t_1, t_2 \geq t_{CLK}$
[Flash mode display or while blanking]



c. Flashless Mode



VIDEO TIMING

All GDC applications require calculation of the parameters for the GDC's video sync generator. These parameters control the various intervals that form the video raster-scan sequence. Factors to be considered when selecting these interval widths include the CRT monitor's characteristics, the desired display format, and the capabilities of the GDC itself. These factors are interactive and are made compatible via the video sync generator parameters and the choice of the clock frequency.

The GDC video sync generator counts out the video format using the basic display memory cycle period as the basic unit of time. Each display cycle uses two cycles of the 2xWCLK clock. This period of time is used for several purposes by the GDC, including active display cycles, blanked cycles, and one-half of a RMW cycle. Zoom magnified display cycles are stretched by multiples of this interval.

Every horizontal line in the video raster is formed by four intervals, each composed of an integral number of display cycle periods. The first is the horizontal front porch (HFP). The next is the horizontal sync pulse (HS), and then comes the horizontal back porch (HBP). The video during these first three intervals is blanked while the CRT display unit horizontally retraces. The fourth interval is the active display period (AW), where the contents of display memory are sent to the screen during active display lines.

The video field is formed by four groups of lines which correspond to the horizontal intervals. They are the vertical front porch, VFP, the vertical sync (VS), the vertical back porch (VBP), and the active lines (AL). All are entirely blanked, except for the active lines which are unblanked during their AW interval. Each vertical interval is generated using an integral number of lines.

A video frame, when generating interlaced video, is composed of two video fields in sequence, each scanning only half of the total data to be displayed. One field scans the even displayed lines and the next field scans the odd lines. In this way, both the horizontal scan frequency and the video data rate are reduced to one-half of their non-interlaced rates.

Parameters Calculations

The sync generator parameters can be specified to generate almost any video sync format when used with the correct clock frequency. Given the initial details of the desired video format and the operational requirements of the CRT display unit, a 10-step procedure is outlined below to find the proper parameters and clock frequency. The assumption is made that the clock frequency is constant. Note that for coded character and character generator applications, a single character replaces the word of 16 pixels used in graphics bit-mapped applications. Mixed mode operation requires a more sophisticated clocking scheme, as does the generation of NTSC composite color signals.

Step 1. Specify the desired display format based on the applications requirements. The visible image on the screen appears during the "active" intervals of the video raster sequence. The active intervals are the unblanked times during which video information is sent to the CRT screen. The entire video sequence will also include blanked intervals to provide time for the CRT display to retrace its beam at the end of each horizontal line and at the bottom of the screen. One of the goals of these calculations is to determine the length of these blanked intervals. Of course, after going through the following calculations, it may be necessary to adjust the display format to meet either GDC or CRT display constraints.

- The number of pixels to be actively displayed horizontally on each raster line (AP) must be selected as a multiple of 32. The GDC limits this to a maximum of 4096 pixels or, in character mode, 256 characters.
- The number of active video lines to be displayed (AL) can be any integer up to 1024 lines per field. If 2:1 interlaced video is used, 2048 lines can be displayed with 1024 lines in each field.
- The speed at which the CRT beam makes one pass from the top of the screen to the bottom must be selected. This determines the video field rate. It is common to use a 60-Hz field rate but almost any other value can be generated by the GDC. United States broadcast television standards call for 59.95-Hz rate. European video systems are based on 50-Hz field rate. High resolution displays might use 40-Hz or lower rates.
- Two design considerations must be taken into account when making this choice. The refresh rate must be high enough to avoid display flicker with the CRT phosphor used, and the upper frequency limitations of the various system components must not be exceeded. The following definitions summarize these relationships:

f_{VERT} = frequency of the vertical sync pulses
t_{FIELD} = time between vertical pulses
= time of a video field
$f_{VERT} = 1/t_{FIELD}$

The GDC can generate interlaced video in which one-half of the display lines are scanned during each video field. Two successive video fields scan the entire image

raster in such a way that the even-numbered lines are scanned in one field and the odd-numbered lines are scanned in the next field. The sync generator parameters control a single field's composition. Interlaced video uses two of these fields plus one line (which is automatically added if interlace operation is specified) to display one video frame (the video entity which scans all the information to be displayed). If interlaced operation is used, the number of displayed lines per field will equal one-half of the lines displayed in total.

DL = total lines displayed
AL = active lines displayed per video field
Video Field = interval between successive vertical sync pulses
Video Frame = interval required to bring all the displayed data to the screen

For interlaced video: AL = DL/2
For non-interlaced video: AL = DL

Step 2. The next step is to determine the requirements of the video CRT unit. The values listed below are important in the calculations establishing the video parameters. The display unit must be capable of displaying the desired video format. It is important during the calculations to check the results against the capabilities of the equipment and devices to find any incompatibilities.

- Vertical blanking requirement, t_{VB} (min and max)
  RS-343: 1250 μs ± 150 μs
- Horizontal blanking requirement, t_{HB} (min and max)
  RS-343: 7 μs ± 0.25 μs
- Horizontal sweep frequency, f_{HORIZ} (min and max)

Step 3. The video field must be composed of an integral number of lines. These lines must be broken down first into the active display lines and the blanked vertical retrace lines. Later, the blanked lines will be further divided into the VFP, VS, and VBP intervals. The number of active lines has been specified above as AL. Here the number of blanked lines must be determined to provide the CRT enough blanking. The sum of the two line types will be the number of lines in the video field and will be used to calculate the horizontal sweep frequency.

To find the number of lines needed for the vertical blanking period it is necessary to find the maximum time of the active interval. Dividing this by the known number of active lines will give the maximum length line time since the overriding concern is to have the lowest horizontal frequency consistent with the CRT's requirements. All lines must be of the same length so we can divide the minimum vertical blanking time by this longest line time to find the number of lines needed for blanking. Of course, it is highly unlikely that this will be an even number. Therefore, it will be necessary to round this up to the next larger integer. This will boost the horizontal sweep frequency slightly above the lowest possible, but it will assure the proper amount of blanking. Knowing the actual number of lines in the raster will allow the calculation of the exact line time in the next step.

In some situations, the total number of lines in the raster, blanked and unblanked, will be given. In this case, the line time will be easy to calculate without the preliminaries described above.

- Vertical active display time, t_{VA}:
  $t_{VA} = t_{FIELD} - t_{VB}$
  t_{VA} (max) $= t_{FIELD} - t_{VB}$ (min)
- Time of one horizontal sweep, t_{LINE} (approximate value):
  t_{LINE} (approx) $= t_{VA}$ (max)/AL
- Number of blanked lines, BL:
  BL = t_{VB} (min)/t_{LINE} (approximate; round up)
- Number of total lines in the video field raster, TL:
  TL = AL + BL

Step 4. Once the total number of lines in the video field TL is determined, the time of a horizontal sweep can be easily calculated using the duration of the video field, t_{FIELD}, which was determined in step 3. The horizontal sweep frequency can then be calculated and compared to the capabilities of the CRT unit.

- Actual horizontal sweep period, t_{LINE}:
  $t_{LINE} = t_{FIELD}/TL$
- Actual horizontal sweep frequency, f_{HORIZ}:
  $f_{HORIZ} = 1/t_{LINE}$

Step 5. Given the exact line time, t_{LINE}, and the number of blanked lines during vertical retrace, the length of the vertical blanking interval can be calculated and compared with the CRT unit's requirements.

- Actual time of the vertical blanking interval, t_{VB}:
  t_{VB} (actual) $= t_{FIELD} - (BL \times t_{LINE})$

Step 6. Now that the major vertical parameters are established, the variables within the video line can be computed. Like the vertical format, each horizontal line can be divided into a blanked retrace part and an active part. The horizontal retrace of the CRT beam is done during the blanked time of each line. The active part of the line is unblanked during active display lines to display the video data on the CRT screen. During the vertical retrace period, this part of the line is blanked along with the horizontal retrace period.

- Maximum time of the horizontal active period, t_{HA} (max):
  t_{HA} (max) = t_{LINE} − t_{HB} (min)
- Active words displayed per line, AW:
  AW = AP/16
- Approximate time of a word display cycle, t_{DC}:
  t_{DC} (approx) = t_{HA} (max)/AW
- Number of blanked display words per line, BW:
  BW = t_{HB} (min)/t_{DC} (approximate; round up)
- Total word accesses per line, TW:
  TW = AW + BW
- Actual time of each display access cycle, t_{DC}:
  t_{DC} = t_{LINE}/TW

Step 7. With the exact length of the basic access cycle time determined, the actual length of the horizontal blanking interval can be computed. The calculations above will yield a value that is a fraction of an access cycle time longer than the target length. Of course, another target time could have been used in place of the minimum CRT blanking requirement if there had been overriding considerations. In either case it is important to find the exact blanking interval time and compare it to the original requirement. In cases of a very narrow range of acceptable values, it may be difficult to find an integral number of cycles that will work.

- Actual duration of horizontal blanking interval, t_{HB}:
  t_{HB} = t_{DC} x BW

Step 8. In the normal graphics operating mode of the GDC, a display cycle accesses 16 pixels to send to the CRT. Therefore, the pixel clock frequency must be 16 times the display cycle frequency, f_{DC}, and eight times the frequency of 2xWCLK. This is the mode of operation used throughout this discussion.

Two other modes of operation can be used instead of the normal 16 pixels-per-word graphics mode. In the wide display access mode, the pixel clock is 32 times the display cycle frequency (16 times f_{2xWCLK}) so that 32 pixels can be accessed from even and odd word pairs simultaneously. In the mixed mode, a graphics area uses four 2xWCLK cycles for each active display cycle. Each display cycle accesses 16 pixels in twice as many GDC clock cycles as the normal mode uses. Therefore, the pixel clock frequency is four times f_{2xWCLK}, instead of the normal eight times. The GDC outputs the display address twice for each display word, effectively accessing each location twice, so the AW parameter value must be twice the normal mode's value for the same number of displayed pixels. During horizontal retrace, two GDC clocks define a "display cycle" word time as in the normal operating mode. If these modes are used, the active words parameter (AW) must be adjusted accordingly for the earlier calculation, as must the relationships between the various clocks.

- Pixel period, t_{PIX} (normal graphics mode):
  t_{PIX} = t_{DC}/16
- Pixel clock frequency, f_{PIX}:
  f_{PIX} = 1/t_{PIX}
- GDC clock period, t_{2xWCLK}:
  t_{2xWCLK} = t_{DC}/2 (two 2xWCLK cycles per display cycle)
- GDC clock frequency, f_{2xWCLK}:
  f_{2xWCLK} = 1/ t_{2xWCLK}

Be sure to compare f_{2xWCLK} to the GDC's upper and lower clock frequency limits. Proper operation of the device cannot be guaranteed outside these limits, even though the device may appear to work under laboratory conditions. If a limit is exceeded, it may be possible to change the mode of operation of the device as described above to move the clock frequency back within the allowed range.

Step 9. The horizontal blanked display cycle words, specified by BW, must be broken down into three intervals for the GDC: horizontal front porch (HFP), horizontal sync (HS), and horizontal back porch (HBP). The relationship between them is:

$$BW = HFP + HS + HBP$$

where each variable is an integer number of display cycle word periods (one display cycle = two 2xWCLK cycles.)

These individual values are chosen based on several considerations. First, there are minimum width requirements for these intervals in various operating modes to provide time for the GDC's command processor to handle its tasks during horizontal blanking time. Second, if dynamic RAM refresh is enabled for the display memory, horizontal sync must be wide enough to provide at least the minimum RAM refresh rate at the horizontal sweep frequency in use. During horizontal sync, each display cycle address comes from an internal refresh counter, if refresh is enabled. These refresh cycles have the highest priority in the GDC's display memory cycle arbitration scheme to ensure they are never interrupted or delayed.

Third, the horizontal sync pulse width and position must meet the CRT display unit's requirements. When there is a conflict between these considerations they must be resolved in the order in which they are presented here. External hardware may have to be added in some cases, or the clock to the GDC can be speeded up during retrace time, etc.

Horizontal front porch (HFP) constraints:

- In general, HFP ≥ 2 display cycles
- If video sync slave mode is used, HFP ≥ 4 display cycles
- If the light pen is used, HFP ≥ 6 display cycles
- If DMA, zoom, or interlace is used, HFP ≥ 3 display cycles

Horizontal sync (HS) constraints:

- If DRAM refresh is enabled, HS ≥ 2 display cycles
- If interlaced display mode is used, HS ≥ 5 display cycles

Horizontal back porch (HBP) constraints:

- In general, HBP ≥ 3 display cycles
- If the IMAGE or WD modes change in the video field, HBP ≥ 5 display cycles
- If the interlaced display and split screen are used, HBP ≥ 5 display cycles

The dynamic RAM refresh rate must be set just fast enough to meet the requirements of the RAMs and to provide as much drawing time during the horizontal line as possible. Considering μPD4164-type 64K RAMs, 128 refresh cycles must be done every 2000 μs. Since refresh cycles are done only during the horizontal sync (HS) interval of each line, each video line must do enough refresh cycles so that in 2000 μs, 128 cycles have been executed. Other RAMs might require a different number of refresh cycles in a different time period.

The variables of concern are:

- Refresh interval, $t_{REFRESH}$ (2000 μs for 4164s)
- Refresh cycle count, RCC (128 for 4164s)
- Time of a horizontal sweep, t_{LINE}
- Number of video sweep lines per refresh interval, LRI:
  LRI = $t_{REFRESH}/t_{LINE}$ (round down)
- Minimum number of refresh cycles per line, RCL:
  RCL (min) = RCC/LRI (round up)
- Number of display cycles in horizontal sync, HS:
  HS ≥ RCL (min)

Step 10. Like the horizontal retrace interval, the vertical retrace interval must also be separated into three parts. But, unlike the horizontal retrace interval, there are no restrictions on the number of lines in each interval except that each must be at least one line long. The three interval line counts must total the number of total blanked retrace lines calculated previously.

BL = VFP + VS + VBP

where: BL = lines in vertical blanking interval
VFP = lines in vertical front porch
VS = lines in vertical sync pulse
VBP = lines in vertical back porch

As a point of comparison, television broadcast standards require VFP = 3, VS = 3, and VBP = BL − VFP − VS. (VFP and the first 3 lines of VBP are used for equalization pulses.)

Monitor Timing Calculations

Here are the crystal frequency and sync parameters for one particular monitor, the NEC CU-2002P2.

Basic monitor timing specifications:

Horizontal Freq	=	22.7 kHz = 44.053 μs = H
Vertical Freq	=	43 to 50.7 Hz, use 50 Hz = 20,000 μs
t_{HB}	=	9.0 μs min, 12 μs typ
t_{VB}	=	1000 μs min, 2112 μs typ
HFP	=	4 μs typ
HS	=	2 μs typ
HBP	=	6 μs typ
VFP	=	528 μs typ (12H)
VS	=	528 μs typ (12H)
VBP	=	1,056 μs typ (24H)
t_{HA}	=	32 μs typ

Number of sweeps in one field (use 50 Hz):

(22.7 kHz sweeps/sec)÷(50 Hz fields/sec)
= 454 sweeps/field

Number of sweeps during t_{VB} = 12 + 12 + 24 = 48 lines

Number of active lines = 454 − 48 = 406 lines
For a 4:3 aspect ratio (AR): 406 x 4/3 pixels/line = 539.98 lines

The GDC must have an even multiple of 16 pixels active:

539.98/16 = 33.75 words

The GDC can generate 32 or 34 words for 512 pixels or 544 pixels. Use 544 pixels by 406 lines for AR = 544/406 = 1.34

Determine the number of word display times per line:

t_{HA} (typ) = 32 μs
t_{DC} = 32 μs/34 words = 941.1764 ns (approx)
t_{HB} (typ) = 12 μs/0.941 μs = 12.75 words (rounded up to 13 words)

Words/line total = 34 active + 13 blanked = 47 words
t_{DC} = 44.053 μs/47 = 937.294 ns
t_{HB} = 13H = 12.185 μs
t_{2xWCLK} = (1/2) t_{DC} = 468.6 ns
t_{PIXEL} = (1/8) t_{2xWCLK} = 58.6 ns
f_{PIXEL} = 17.070439 MHz ←XTAL OSC
f_{2xWCLK} = 2.1338 MHz

Horizontal blanking interval:

13 words t_{HB} = HFP + HS + HBP

t_{DC} = 937.3 μs
HFP = 4 μs
HS = 2 μs
HBP = 6 μs
HFP → 4 μs/t_{DC} = 4.26 words (3 lines) → 4 words (3 lines) = 3.75 μs
HS → 2 μs/t_{DC} = 2.13 words (3 lines) → 3 words (3 lines) = 2.81 μs
HBP → 6 μs/t_{DC} = 6.40 words (3 lines) → 6 words (3 lines) = 5.62 μs

Vertical blanking interval:

t_{VB} = 48 lines = VFP + VS + VBP
VFP = 12 Lines
VS = 12 Lines
VBP = 24 Lines

Video Format Check

XTAL OSC frequency = 17.070439 MHz

Number of words per line = 47
Number of lines per field = 454

t_{DC} = 16/17.070439 MHz = 937.2928 ns
t_{LINE} = 47 t_{DC} = 44.052761 μs → 22.7 kHz
t_{FIELD} = 454 t_{LINE} = 19999.953 μs → 50 Hz
t_{HB} = 13 t_{DC} = 12.18 μs
t_{VB} = 48 t_{LINE} = 2114.5 μs

HFP = 4 t_{DC} = 3.749 μs
HS = 3 t_{DC} = 2.812 μs
HBP = 6 t_{DC} = 5.624 μs

Interlaced Video

Interlaced video (2:1 interlaced) is generated using a two-video field sequence, in which the total line count for both fields is odd, and the vertical sync pulse between the two fields is offset by exactly one-half line time. These conditions cause the display lines of the second field to be displayed between the lines of the first field. In this way, an image can be displayed without flicker at a reduced bandwidth. Interlacing is particularly suited to the display of camera- (or computer-) generated images (as differentiated from line drawings and text) because of the spatially band-limited nature of these images. This band-limiting exists along both the horizontal sweep line and vertically across the video lines. Interlaced video is less well suited to typical computer-generated graphics and text displays in that there is no spatial band-limiting in the vertical direction. For example, one line in the video raster might be black while the very next line is full white. If these lines are scanned in alternate video fields as they are in using interlaced video, there will be very noticeable flicker and apparent vibration. Long persistence phosphors can minimize the problem.

The GDC can be programmed to generate interlaced video. The separate horizontal and vertical sync signals can be used with any CRT unit which accepts separate sync signals. The total line count of both fields, one video frame, will be one more than twice the line count per field for which the GDC is programmed. The GDC's video sync generator adds this extra line automatically and offsets the second VSYNC pulse by approximately one-half line. In typical applications, the second field will be displayed within 15% of the ideal position between the lines of the first field.

Transitions of the VSYNC output during interlaced operation take place at two different points with respect to the HSYNC pulse. First field transitions (both leading and falling edges) occur simultaneously with the leading edge of BLANK. In the second field, the transitions occur three 2xWCLK cycles before the middle of the active words interval, AW. It does not automatically position the VSYNC transitions at the exact midpoint of the line including the retrace blanking period. The exact relationship is shown in figure 6-1 and the following.

Interval A = 2 (HFP + HS + HBP + AW/2) − 3 (2xWCLK cycles)

Interval B = 2 (AW/2) + 3 (2xWCLK cycles)

Horizontal blanked display cycles, HB:

HB = HFP + HS + HBP

Total number of 2xWCLK cycles during a horizontal line, TC:

TC = HFP + HS + HBP + AW

The line pairing interlacing error percentage, LPE (percentage of a half line of offset):

LPE = ((interval A − Interval B)(TC/2)) x 100%
= ((2HB + AW − 3) −(AW + 3)/(TC/2)) x 100%
= ((2HB − 6)/(TC/2)) x 100%
= ((4HB − 12)/TC) x 100%

The line pairing error for a typical application:

TC = 100 2xWCLK cycles
HB = (1/5 x TC)/2 blanking percentages of 20%

LPE = ((4 x 10 − 12)/100) x 100% = 28%

The differences between standard interlaced video and the interlaced video signals produced using the 7220 are elaborated below.

Horizontal Sync. In standard video, equalization pulses are generated during the VSYNC period to keep the receiving television or monitor's horizontal oscillator from drifting, and thus keeping the set in sync. The GDC, however, does not generate these pulses explicitly. Equalization pulses independent of standard HS pulses were determined to be unnecessary due to the high quality of monitors built in the last few years.

Line Spacing. During standard interlaced video, VSYNC on odd video fields (1st, 3rd, 5th, etc) begins and ends with the start of the horizontal front porch. VSYNC on even fields, however, begins at the midpoint of the period consisting of the sum of the horizontal front porch, the horizontal sync period, the horizontal back porch, and the active video period for each line.

The GDC knows the values of horizontal front porch (HFP), horizontal back porch (HBP), horizontal sync period (HS), and active video period (AW) in terms of words of display memory. As it does not have the facilities to add these values together, however, it uses another method to generate the timing for the even VSYNC pulses. They are generated three 2xWCLK pulses before the midpoint of AW (figure 6-2).

Figure 6-1. Horizontal Timing Relative To Vertical Sync Transitions

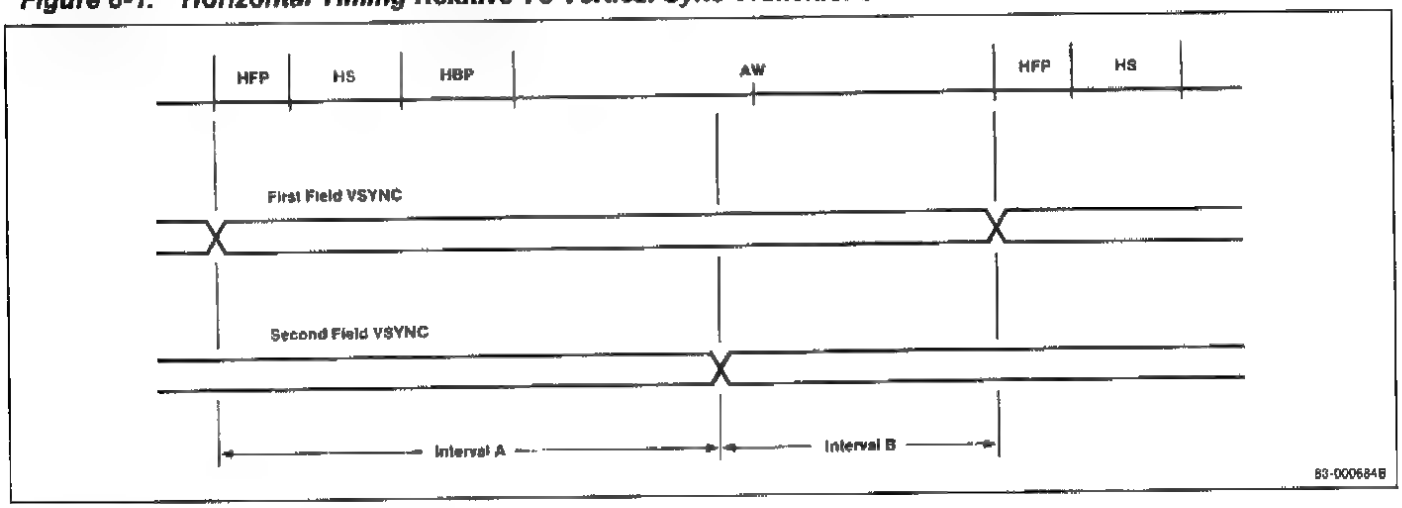
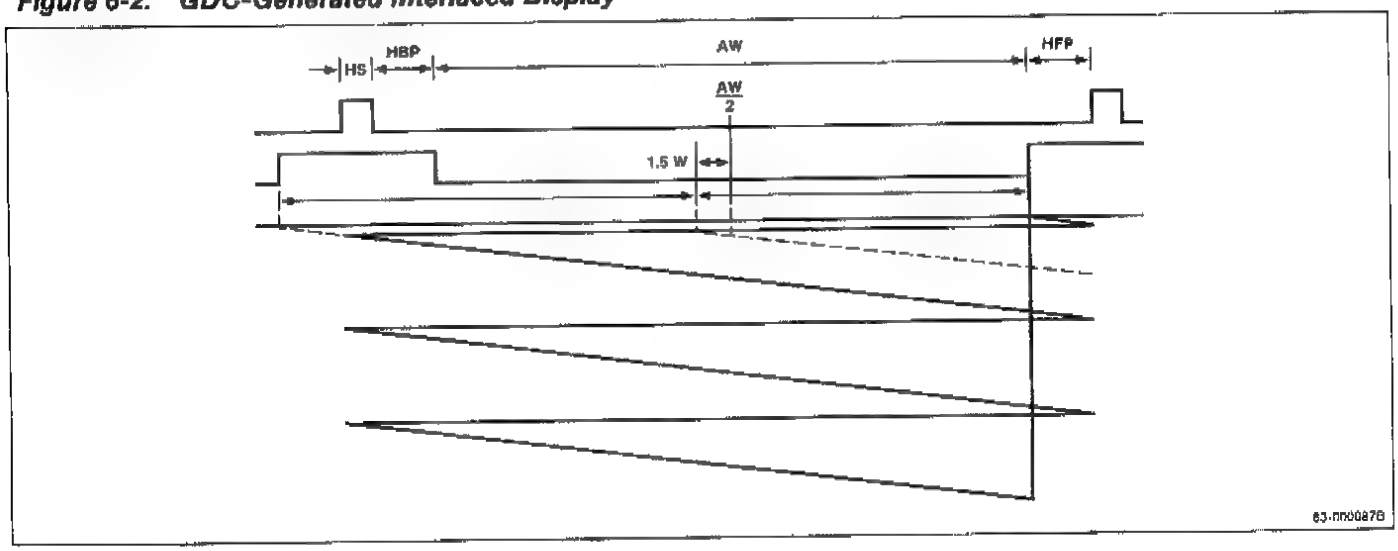


Figure 6-2. GDC-Generated Interlaced Display



The resulting line pairing problem is observed as uneven line spacing between adjacent lines. This problem is only observed on low quality monitors and will not be a problem in most applications. The actual spacing between the lines may be calculated as follows:

Field 1 line − field 2 line = AW/2 + 1.5

Field 2 line − field 1 line = HFP + HS + HBP + AW/2 − 1.5

In most systems, the sum of HFP, HBP, and HS represents 25% or less of one horizontal line period. Therefore, the deviation between the two line spacings will be less than 15%. This difference is not noticeable in the vast majority of designs.

Systems in which this spacing difference is a problem may use the circuit illustrated by figures 6-3 and 6-4 to correct the VSYNC generation.

CURSOR DISPLAY

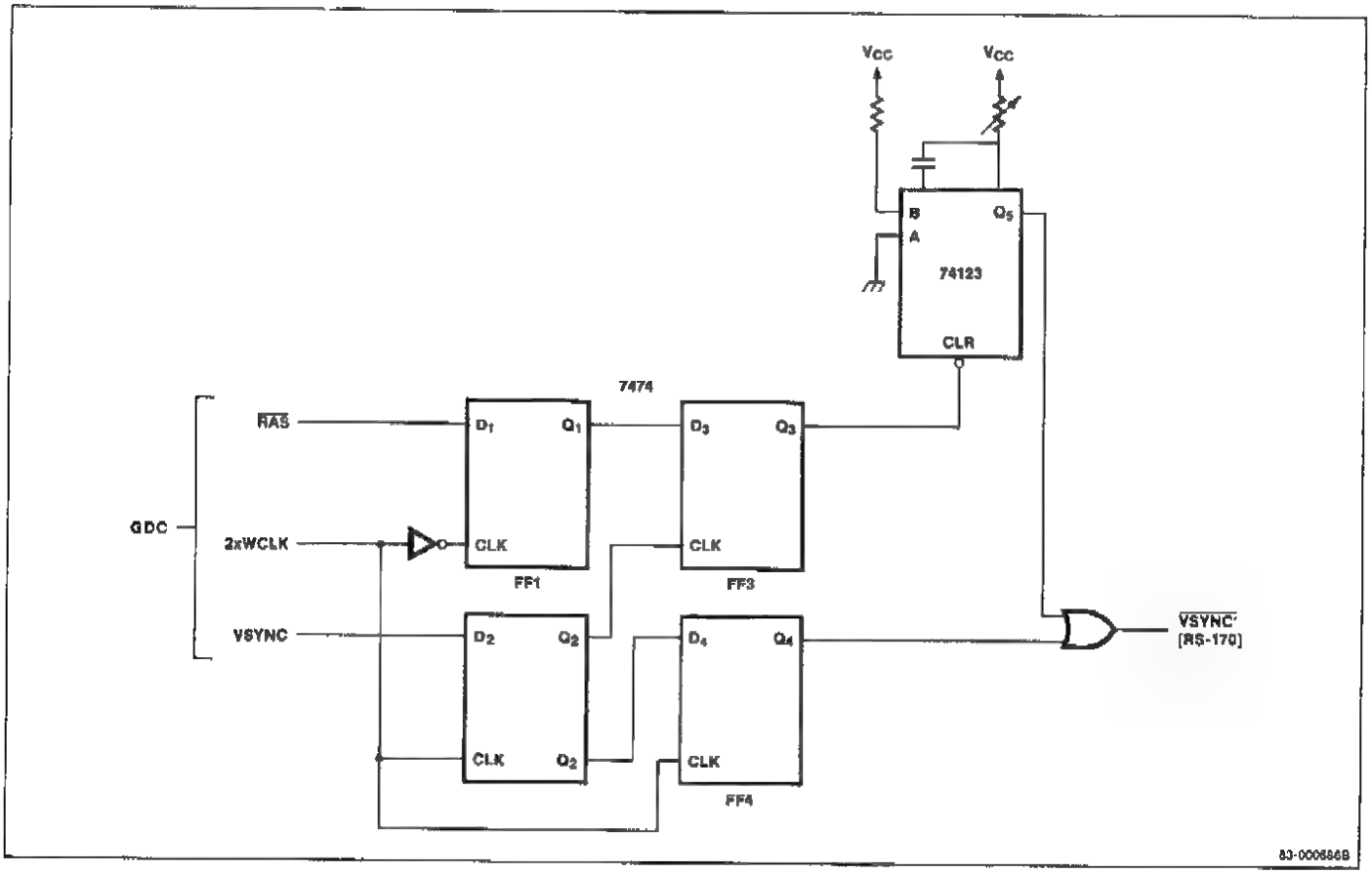
A cursor display indication is generated on pin 39 (A_{17}) of the GDC during the active display intervals of character mode and the character areas of mixed mode. The output is high during the display access time of the word pointed to by the EAD word address of the cursor. This signal can then be used by the external video circuitry to indicate the cursor on the screen with reverse video, a blinking underline, etc.

Wide Display Mode

The operation of the cursor becomes more involved when the wide display access mode is used. In this situation, the GDC is incrementing the display address, DAD, by two for each display cycle. The cursor indication during the display of coded characters must take this into account. The GDC outputs the cursor display indication in the appropriate half of the display cycle in which the character is accessed.

If the starting address, SAD, of the display area partition is even, the GDC will generate only even addresses as it advances by two addresses for each display cycle. The first half of the display cycle outputs the cursor display status of the first character to be serialized, which in this case is an even address character. Of course, the GDC expects the external video hardware simultaneously to access the character at the next-higher odd address. The signal level of the cursor pin during the second half of the display cycle indicates the cursor status of this character, which occupies the next-higher odd address.

Figure 6-3. VSYNC Correction Circuit



Note that the video hardware is assumed to display the even address character first, and then the odd address character at the next-higher address in display memory. The second principle is that the cursor indication for a given character is output one display cycle before the half-cycle in which the character will be displayed. If the cursor signal is shifted through two bits of delay, clocked by 2xWCLK rising edges, the resulting cursor signal will line up with the display time of the character.

If the starting address of the display area is odd, the first display cycle will use the odd address as the DAD to the display memory, but the GDC assumes that the video shift registers still serialize the characters in the even-address-then-odd-address manner. In other words, the second half of the display cycle still corresponds to the odd address character. The difference is that the GDC will be generating the address of the odd address character of the pair. The least significant bit of the display cycle address should be ignored when generating video.

Cursor Programming Considerations

For proper display of other than a block cursor (cursor is the entire row height), the cursor should not be programmed to display on the bottom line of the character row. Under some conditions when the cursor is in the top character row, the CTOP parameter is assumed to be 0 instead of the programmed value, so that the cursor will be displayed from the top line of the character row to the CBOT line. The CTOP parameter is correctly used in the other character rows.

For interlaced video operation, all the parameter bytes must be given during initialization regardless of operating mode. This is true for graphics, mixed, and character modes. For non-interlaced operation in graphics mode, only the first parameter byte need be sent to the GDC.

Figure 6-4. Timing Waveforms for VSYNC Correction Circuit

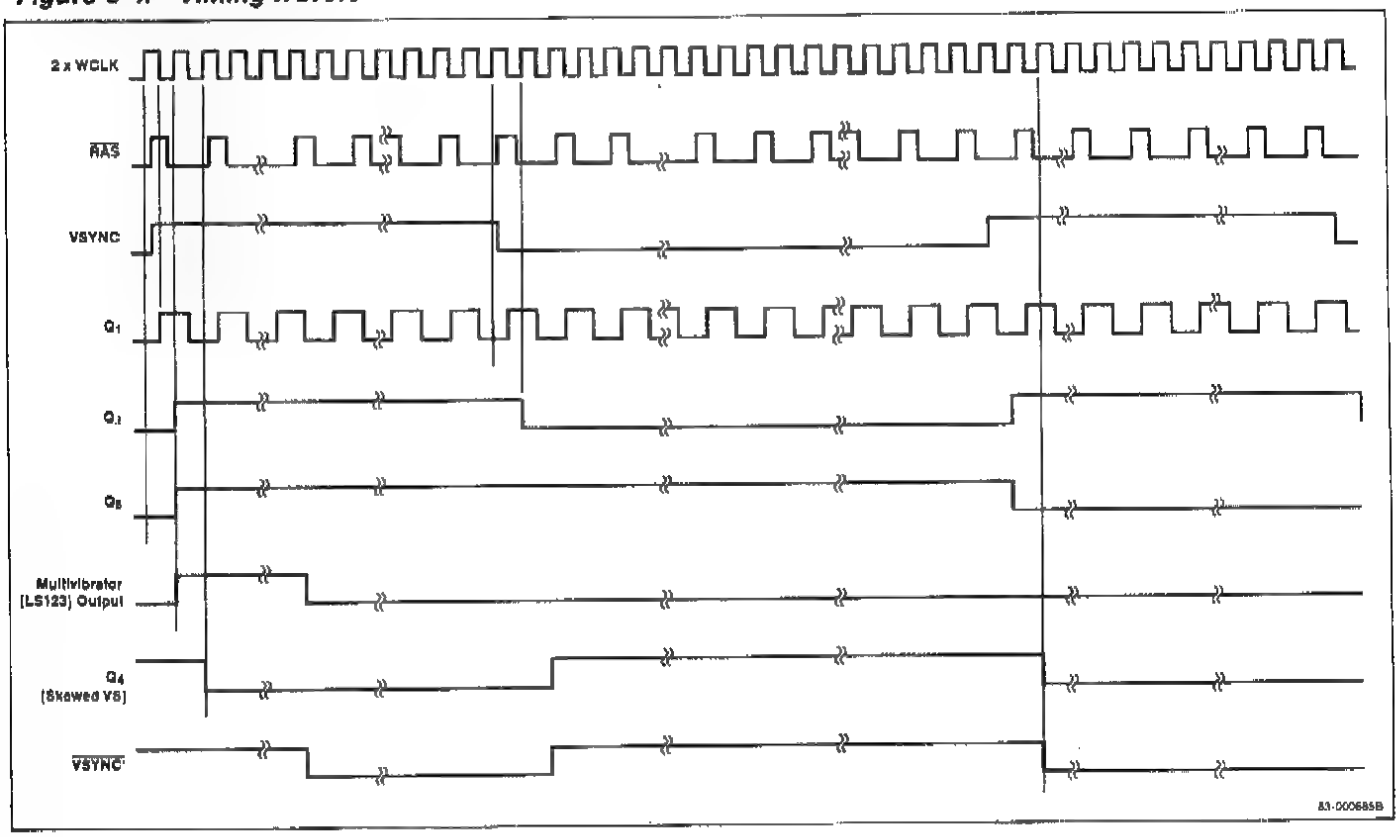


Figure A-1. Wide Display Cycle Memory Configuration

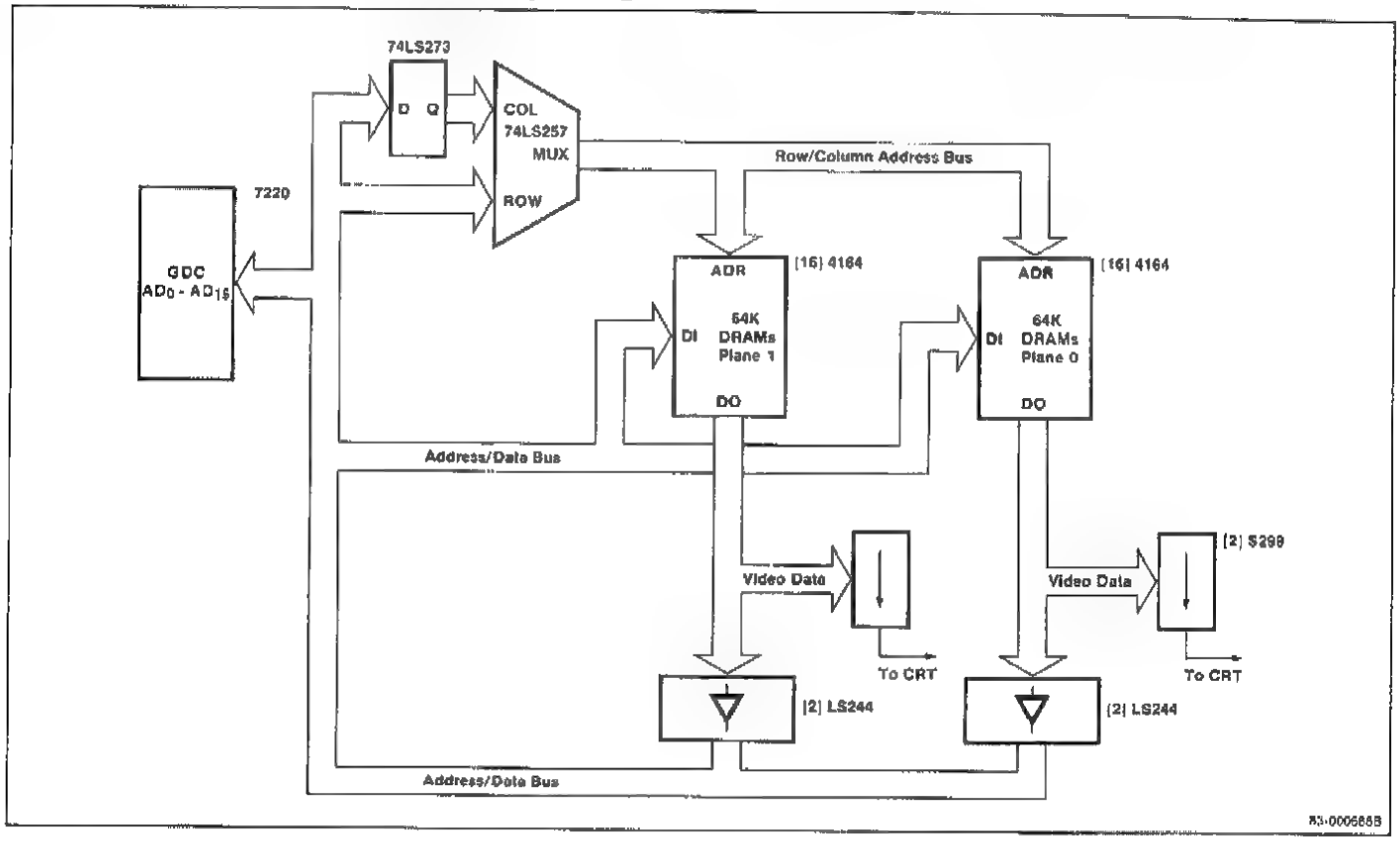


Figure A-2. 4-Plane Display System with Character Mode Option

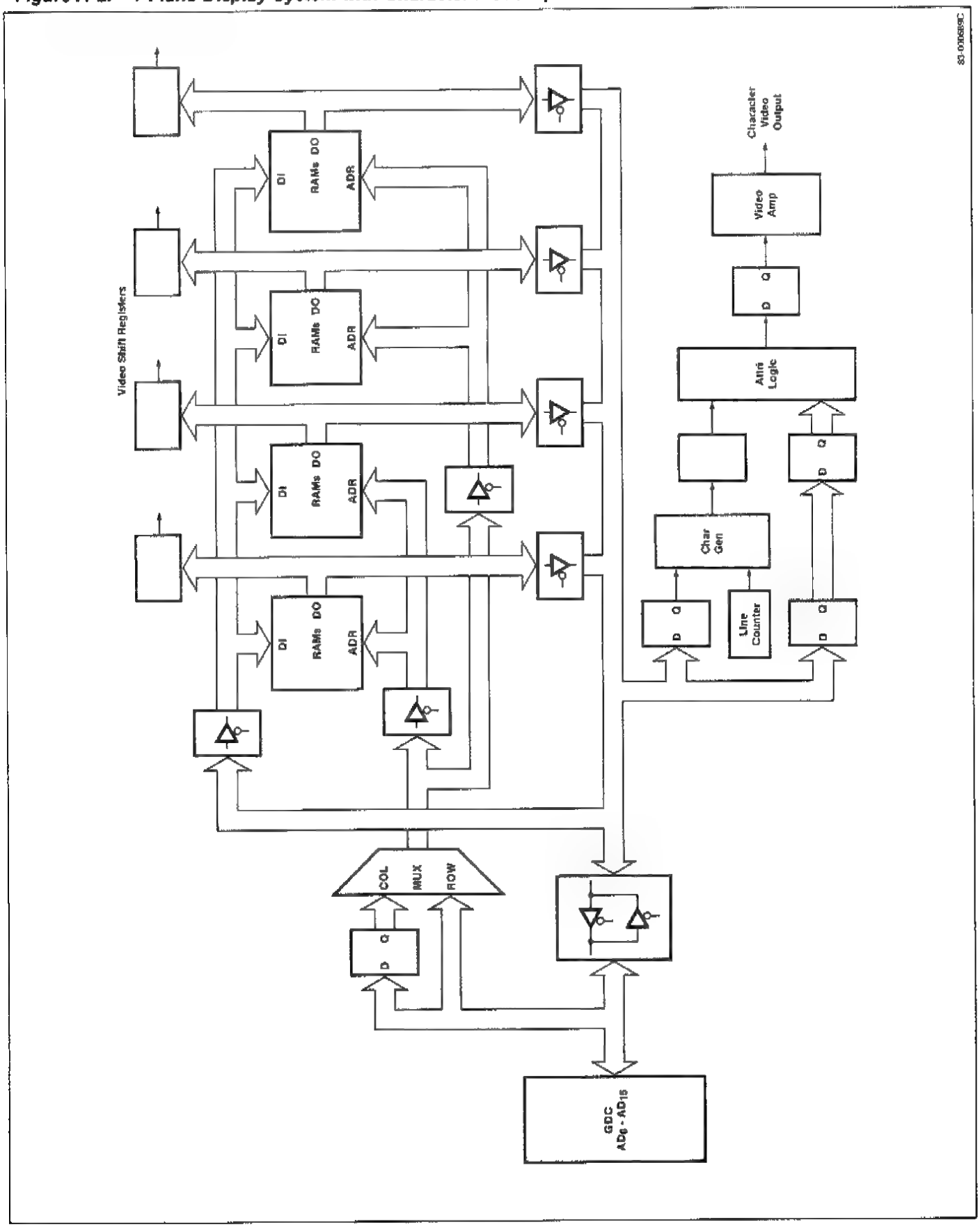


Figure B-1. One-Plane System, Part A

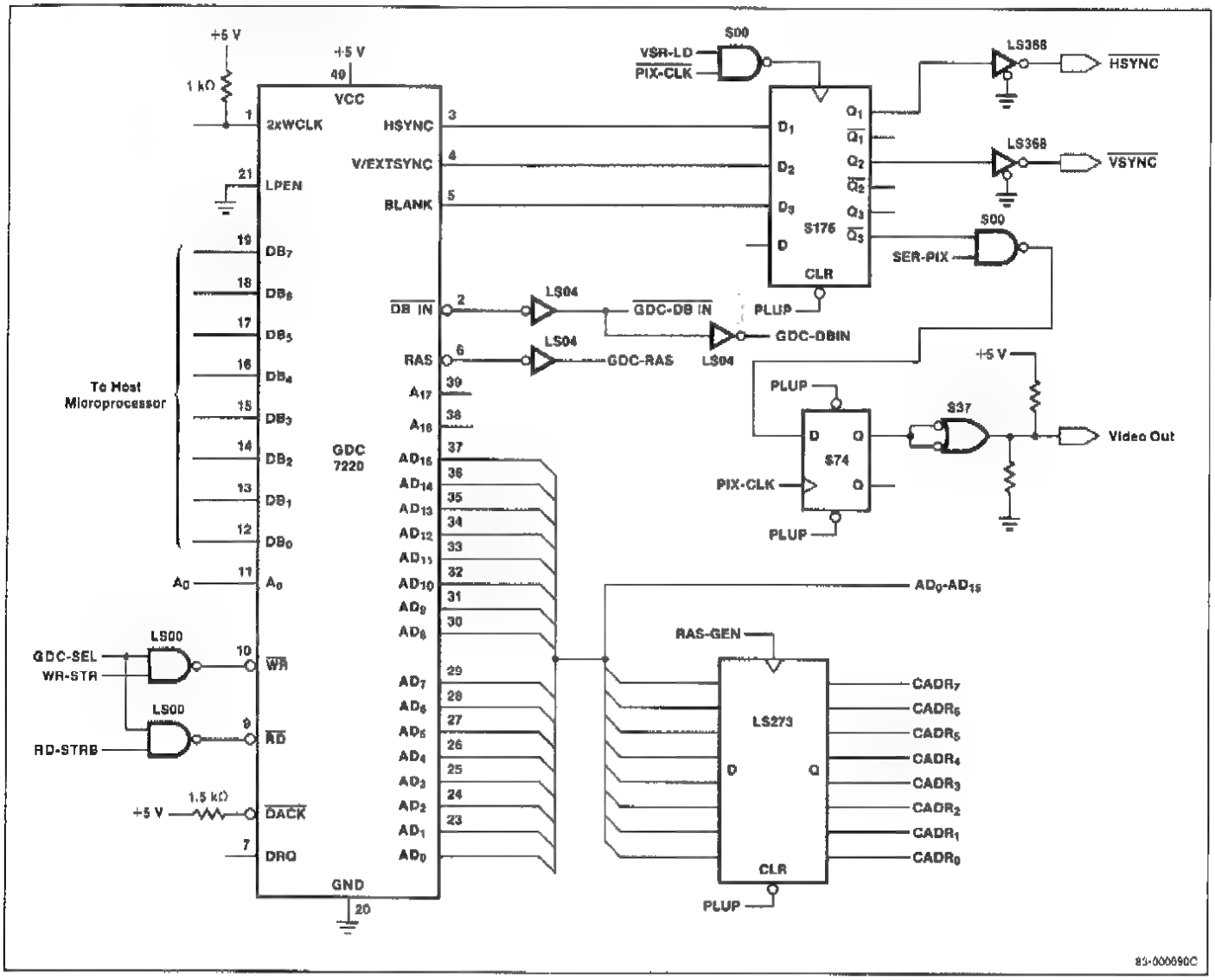


Figure B-2. One-Plane System, Part B

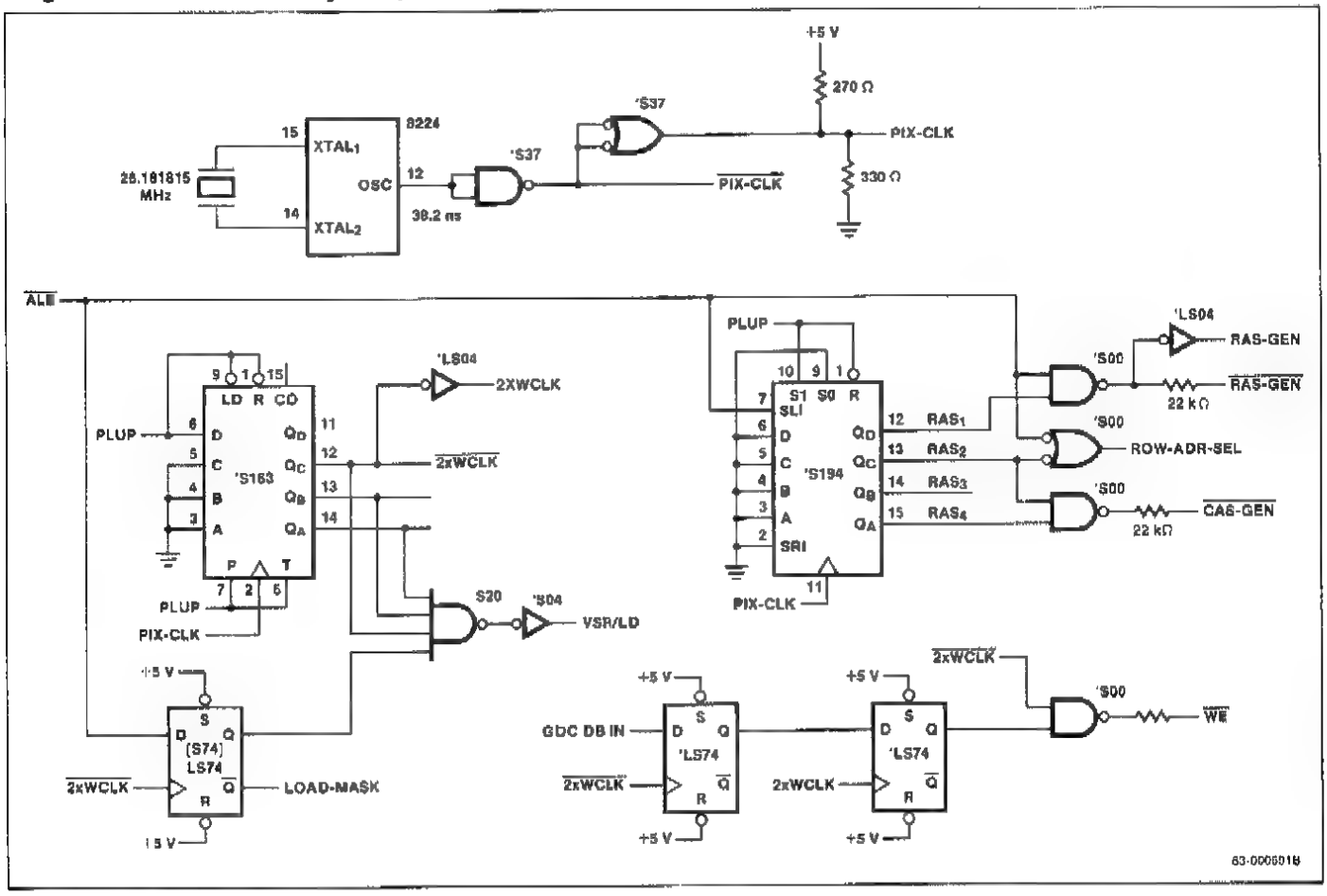
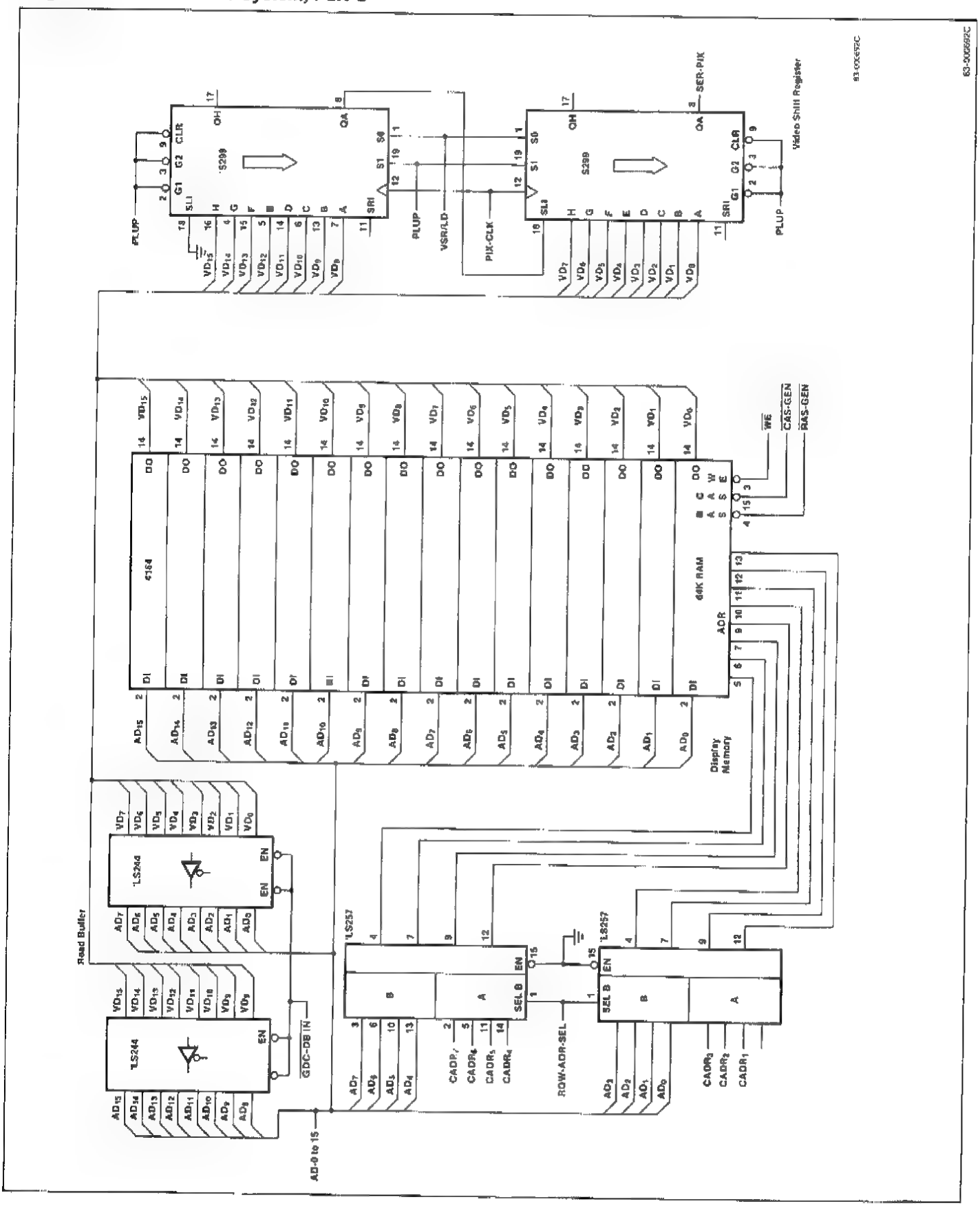


Figure B-3. One-Plane System, Part C



Appendix C
Circle Drawing

A few users have reported difficulties drawing circles with the μPD7220 graphics display controller chip in complement drawing mode. Sometimes gaps occur at pixel locations on the X = Y, the X = -Y, or the X and Y axes. These gaps do not occur for all circles. They depend on the radius of the circle.

Gaps on the X and Y Axes. These gaps are caused by an incorrect dots masked (DM) parameter. The DM parameter sets the number of dots that are not drawn during arc drawing. DM = radius x sin (angle between axis and beginning of arc). If the beginning pixel of an arc is on an axis, the DM parameter is zero and the pixel on that axis is drawn. If another arc begins on the same axis, the pixel on that axis is drawn twice and appears as a gap. To correct this, set DM = 1 for one of the two arcs. Then the pixel will be drawn only once.

Gaps on the X = Y and X = -Y Axes. These gaps are often caused by an incorrect drawing count (DC) parameter. DC is set to one less than the number of RMW cycles to be executed. Consider three categories into which any circle could fall:

Remainder of radius/sin45° < 0.25: In this case, radius/sin45° is rounded down to the nearest integer to obtain the DC parameter. This causes the middle pixels to be redrawn. To correct this, decrement the DC parameter by 1 on alternate arcs.

Remainder of radius/sin45° > 0.75: In this case, radius/sin45° is rounded up to the nearest integer to obtain the DC parameter. This means the middle pixels will not be drawn at all. To correct this, increment the DC parameter by 1 on alternate arcs.

Remainder of radius/sin45° is between 0.25 and 0.75 (inclusive): Radius/sin45° is rounded down to the nearest integer to find the DC parameter. Do not increment or decrement the result, or the pixels at the listed axes will not be drawn at all.

An example of these corrections, written in C, is given in figure C-1

Figure C-1. Complement Mode Circle Correction Program

```
cir2(x,y,rad,color)
int x,y,rad,color;

int dc,d,d2,d1,dm,dir;
int plane,dad;
int testvar;
unsigned cursor;
float value,rem;

value = rad * 0.7071067812 + 0.25;
dc = dt = (int) value;
rem = value -dc;

if (rem < .5) testvar = 1; else testvar = 0;

d = rad -1;
d2 = d * 2;
d1 = -1;
dm = 0;

wdat (COMPLE,WORD,0,0);
for (plane = 0; plane < 3; plane++)

  {
  if (color & (1 << plane))

    {
    for (dir = 0; dir < 8; dir ++

      {
      switch(dir) {
        case 0:
        case 3:
          cursor = ((x-rad) >> 4) + (y << 6);
          dad = (x-rad) & 0x0f;
          curs(cursor,plane,dad);
          break;
        case 1:
        case 6:
          cursor = ((x) >> 4) + (y-rad << 6);
          dad = x & 0x0f;
          curs(cursor,plane,dad);
          break;
        case 2:
        case 5:
          cursor = ((x) >> 4) + (y+rad << 6);
          dad = x & 0x0f;
          curs(cursor,plane,dad);
          break;
        case 4:
        case 7:
          cursor = ((x+rad) >> 4) + (y << 6);
          dad = (x+rad) & 0x0f;
          curs(cursor,plane,dad);
          break;

          }
        if (dir & 0x01){
          if (testvar == 1) -dc;

        }
        else {
          dm = 1;

        }
        figs(ARC,dir,dc,d,d2,d1,dm);
        dc = dt;
        dm = 0;
        figd ();

          }
      }
    }
  }
```

In high to very-high resolution implementations of the μPD7220/7220A, the GDC 2xWCLK (input clock) frequency tends to be at or very near the upper speed limit of the part. The resulting drawing speed, which is directly related to the input clock frequency, is also near the maximum obtainable. As the intended display resolution drops, the physical length of each display memory word as it appears as part of a raster line on the CRT increases. This length increase is observed by the GDC to be an increase in the display cycle period. It follows that the 2xWCLK clock frequency must be decreased to increase the cycle period. The foremost negative side effect of this action is a reduction in drawing speed.

Many of the medium to low resolution implementations of the GDC use a clock frequency between 1.6 MHz and 3.2 MHz. Through the addition of very little circuitry and utilization of the image-bit option of the μPD7220 or 7220A, it is possible to double the input clock frequency without affecting display timing or resolution. What is affected, however, is drawing speed, which is doubled. Before this mode is explained in detail, a brief review of the standard GDC cycle timing follows.

The GDC has two primary types of cycles: read-modify-write (RMW) and display. The RMW is four 2xWCLK clock cycles long while the display cycle is only two 2xWCLK clock cycles long. Figure D-1 shows general timing waveforms for an RMW cycle and a display cycle.

To allow the GDC to draw twice as fast without affecting display timing, either the RMW cycle must be cut in half or the display cycle must be doubled in length. As the former is impossible, it follows that the Image bit accomplishes the latter. Figure D-2 illustrates the effect of invoking this mode.

Figure D-1. Timing Waveforms for RMW and Display Cycles

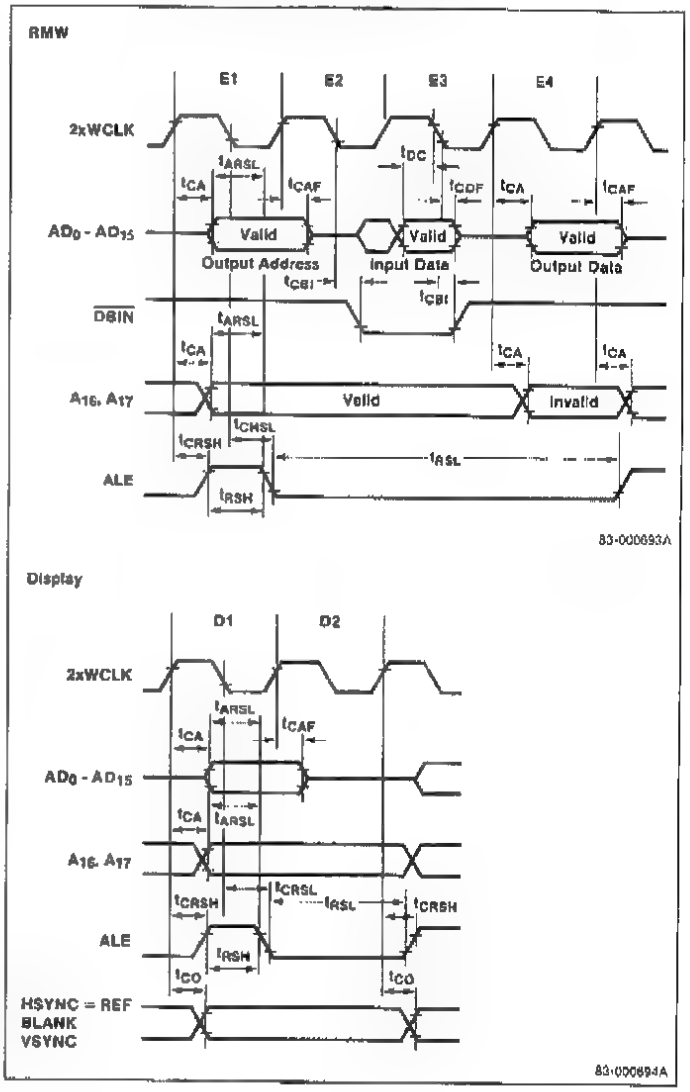
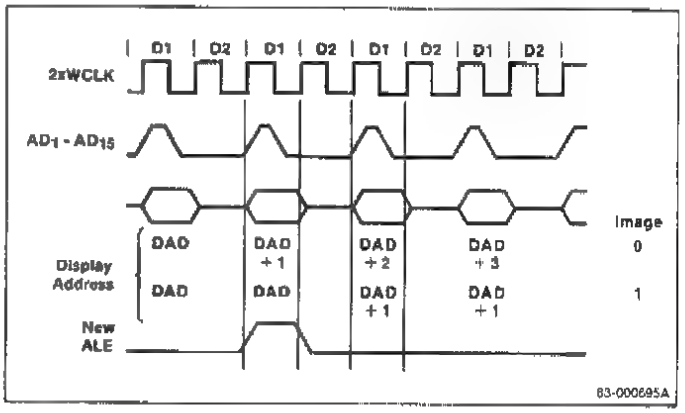


Figure D-2. 2 x Display Cycle



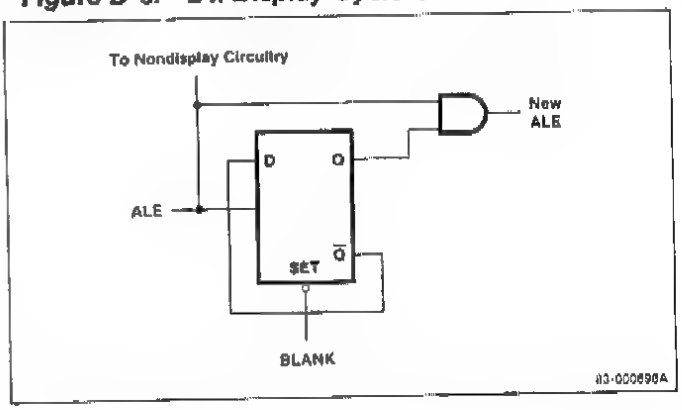
Although it appears that the GDC simply repeats each display cycle (of a given display address) twice by "filtering" out every other ALE pulse, a display cycle appears to other external circuitry as being four 2xWCLK cycles long. An example of the circuitry necessary to accomplish this task is given in figure D-3.

Note that the BLANK signal is also used by this "divide by two" circuit. This is because the filtering of the ALE pulses described above must not occur for any cycle other than display.

The ALEs must be filtered because video shift register load timing is generally generated from this signal. If the signal were not modified in the above manner, a shift register load pulse would occur twice for each display cycle and the first half of each display word would be shifted out twice while the second half would not be shifted out at all.

Once this change has been made to the ALE signal, the only other hardware change required is to double the incoming GDC 2xWCLK clock frequency. In most cases this means dividing the pixel clock by four instead of eight and therefore just changing a tap on a shift register divider.

Figure D-3. 2 x Display Cycle Circuit



The most significant difference between the current μPD7220 graphic display controller and the μPD7220A is the speed increase of the μPD7220A. Currently, the 7220 is available in 4.0-MHz, 5.0-MHz, and 5.5-MHz versions. The 7220A will be available in 6.0-MHz, 7.0-MHz, and 8.0-MHz versions. A number of other differences, in the form of corrections and enhancements, that exist between the parts are described below.

Commands

Two new RESET command variations and one new BLANK command variation have been added to the GDC. All of the new commands allow a reset or display blank to be accomplished while preventing reinitialization of the internal sync generator by an external sync source (slave mode).

	Command	Opcode
RESET1	Same as RESET in 7220	0 0 0 0 0 0 0 0
RESET2	Blank display; no resync	0 0 0 0 0 0 0 1
RESET3	Enable display; no resync	0 0 0 0 1 0 0 1
BLANK1	Same as BLANK in 7220	0 0 0 0 1 1 0 0
BLANK2	Blank display; no resync	0 0 0 0 0 1 0 1

Operation may be characterized as follows:

	Display Blanked	Reset Performed	External Sync (Slave Only)
RESET1	Yes	Yes	Accepted
RESET2	Yes	Yes	Ignored
RESET3	No	Yes	Ignored
BLANK1	Yes	No	Accepted
BLANK2	Yes	No	Ignored

Flag Bits

Six additional operation flag bits have been added. They are located as shown in figure E-1 and their effect on GDC operation follows.

PH Bit. The width of the display pitch register has been increased from 8 to 9 bits. This additional bit (PH) is defined as the seventh bit of the fifth parameter following a SYNC or RESET command. Utilizing this bit, the GDC can handle pitches up to 511 words per line.

DS Bit. A drawing wait function has been added to the GDC. The Light Pen input (LPEN) may be used as a drawing wait control input. If this input is held high for a period greater than four 2xWCLK cycles drawing execution is halted. During the wait cycle, display address (DAD) is output instead of drawing address (EAD). See figure E-2.

Figure E-1. Additional Flag Bits

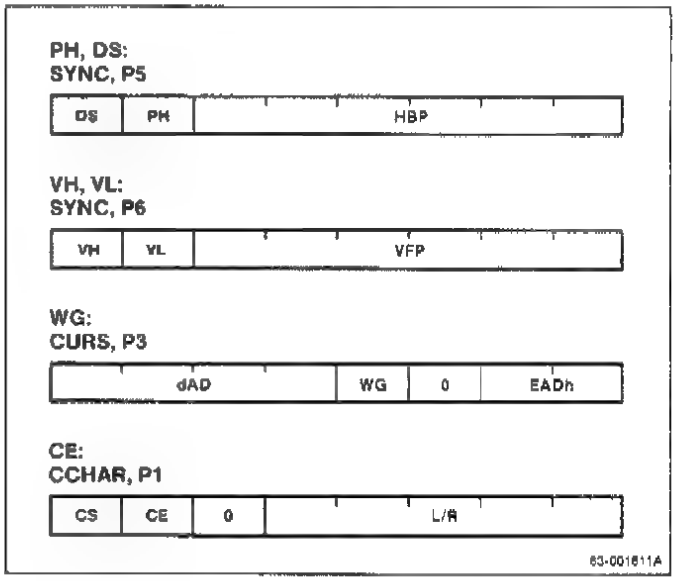
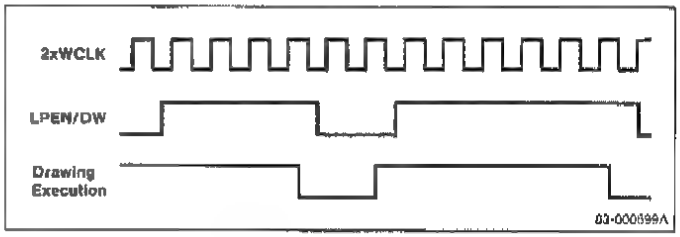


Figure E-2. Draw Wait Timing



VH Bit. A vertical blank status flag is available in the status register to replace the horizontal blank status flag if required.

7220: Outputs a horizontal blank status flag as DB6 in the status register.

7220A: User may select between the standard horizontal blank status flag and the new vertical blank status flag.

Selection is via the new VH flag. This flag is the eighth bit of the sixth parameter following a SYNC or RESET command. When VH is 1, vertical blank status is output. When VH is 0, operation is as in the 7220.

VL Bit. The number of display lines per video frame may be selected as odd or even when in interlaced mode if this bit is set.

7220: Allows only an odd number of lines per video frame.

7220A: Allows an even or odd number of lines per frame. Selection is via the VL flag created in the seventh bit of the sixth parameter byte following a SYNC or RESET command.

WG Bit. When in graphics mode, the state of a newly defined bit in the CURS (cursor position specify) instruction will determine the operation of the WDAT command. This flag bit is known as the WG bit, and is the fourth bit (from LSB) in the third parameter of the CURS command.

7220: In graphics mode, only patterns of 0FFFFH or 0000H may be written with the WDAT command. The pattern written is determined by the least significant bit of each parameter byte following the WDAT command. This bit is expanded into 16 identical bits which form the pattern.

7220A: When the WG bit is set to one, any data following the WDAT command is written as is. If WG is set to zero, the 7220A performs as the 7220 does. The WG bit is not modified by the WDAT or any drawing commands.

CE Bit. Unconditionally reinitializes the internal sync generator when the GDC is programmed for slave mode and the falling edge of an external sync input is detected. When this bit is set, the GDC will resynchronize its vertical sync to the external source during every frame. If the CE bit is set to 0 the GDC ignores any external sync after execution of the start command.

Cursor Position

Cursor postion in character mode is specified by the lower 16 bits of address rather than the lower 13 bits as in the 7220.

Cursor Format

Cursor format restrictions in character mode or character areas in mixed mode have been removed.

Pins A_{16} and A_{17}

A_{16} and A_{17} will now provide static signals.

7220A: A_{16} and A_{17} are invalid during E4 of a read-modify-write cycle. Currently, an external latch is required.

7220: A_{16} and A_{17} will be valid during E4 and any other time. No external latch is required.

Display Word

Display word addresses are incremented during the AW and HPF periods, even during vertical blanking time.

ALE Signal

The ALE signal generated by the 7220A remains as per specifications even while zooming. Corrects problem 3 as described in Product Bulletin #29.

7220: ALE is lengthened by one half 2xWCLK cycle if horizontal blanking begins while in a zoomed display.

7220A: The problem no longer exists.

Resistors

On-chip pull-up/pull-down resistors have been added.

7220: All such resistors are external.

7220A: Added are a pull-up resistor for V/EXT SYNC and $\overline{\text{DACK}}$, and a pull-down resistor for LPEN/DH.

Software

In the area of software, only one modification to existing software is required. Bit 0 of parameter RAM bytes 3 and 7 must be set for the GDC to operate in image mode. These bits are known as the IM or image bits for display partitions one and two. Note that if the above circuitry is installed and these bits are not set, the display will not be correct as every other display word would be skipped.

Figure F-1 shows a design example of a 4-bit wide memory organization. There are three planes of 64K x 4 memory. The display resolution is 512 x 512. This kind of memory organization uses fewer memory chips than the standard 64K x 16 configuration. The major drawback of this method is that it does not support high-resolution display systems. The only software consideration is to set every fourth bit of the MASK register to one after issuing the CURS command. For example, if the drawing is to be started from the first pixel, the mask register then should be configured as below:

MSB															LSB
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0

Figure F-1. Mask Register Bits (Sheet 1 of 3)

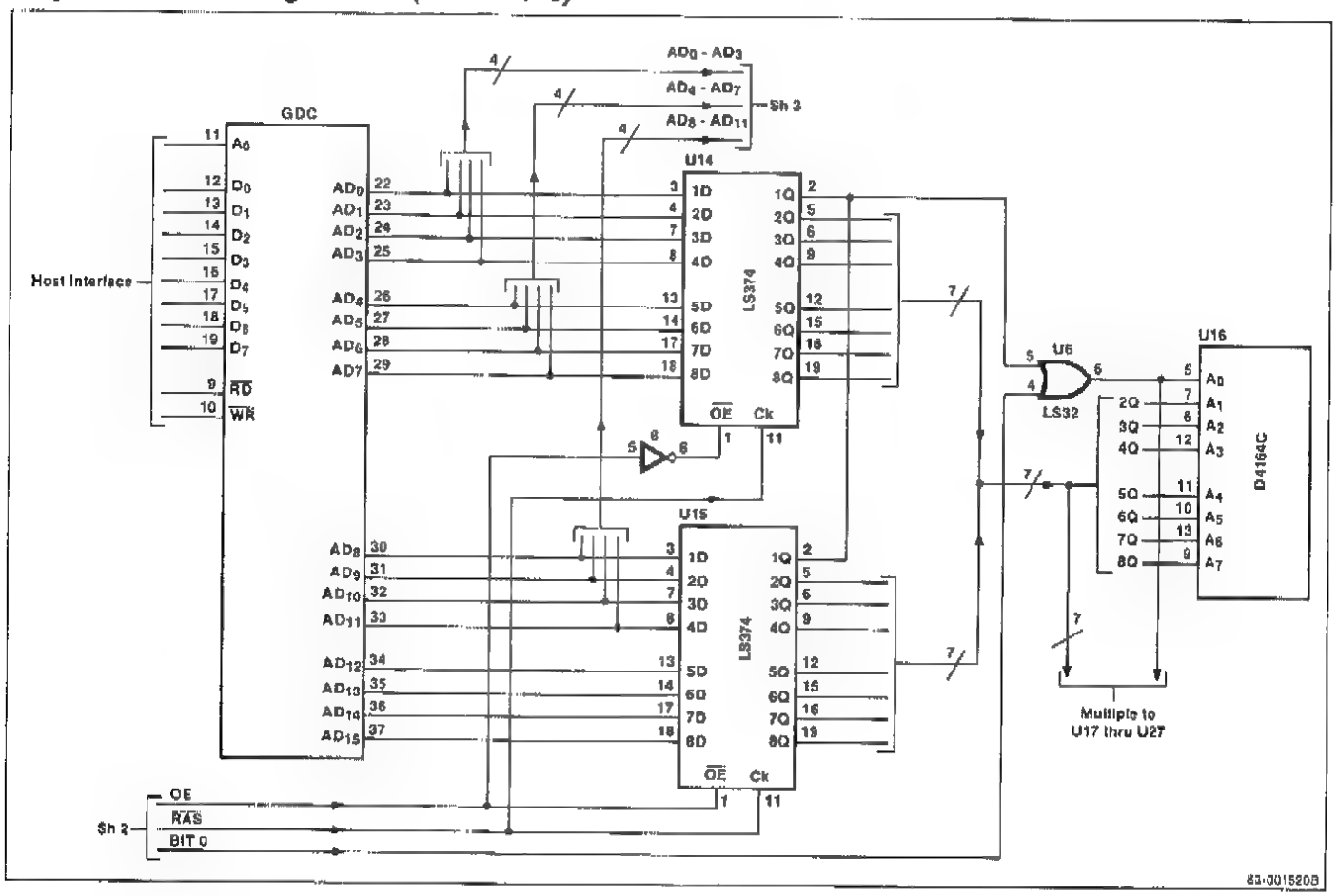


Figure F-1. Mask Register Bits (Sheet 2 of 3)

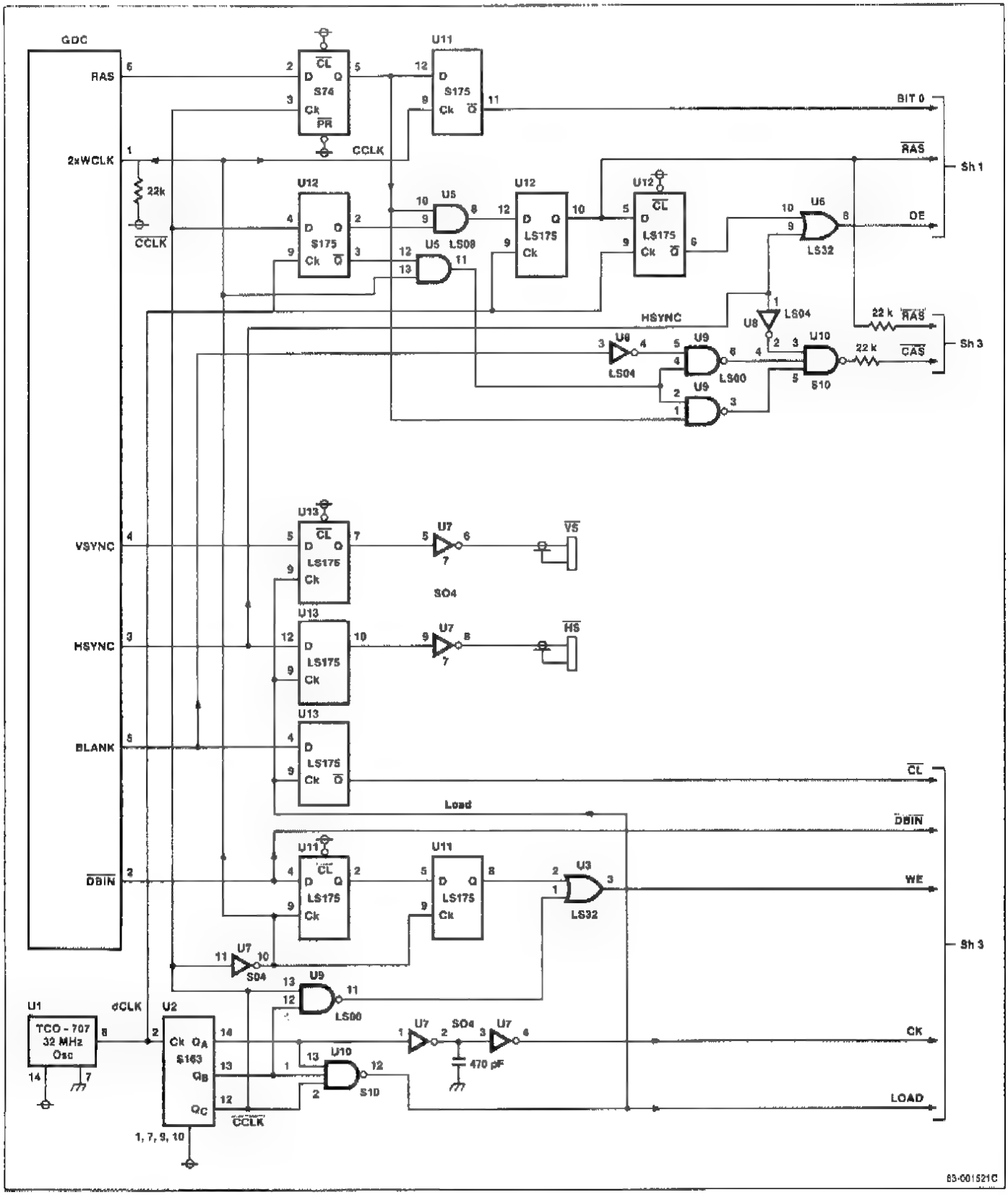
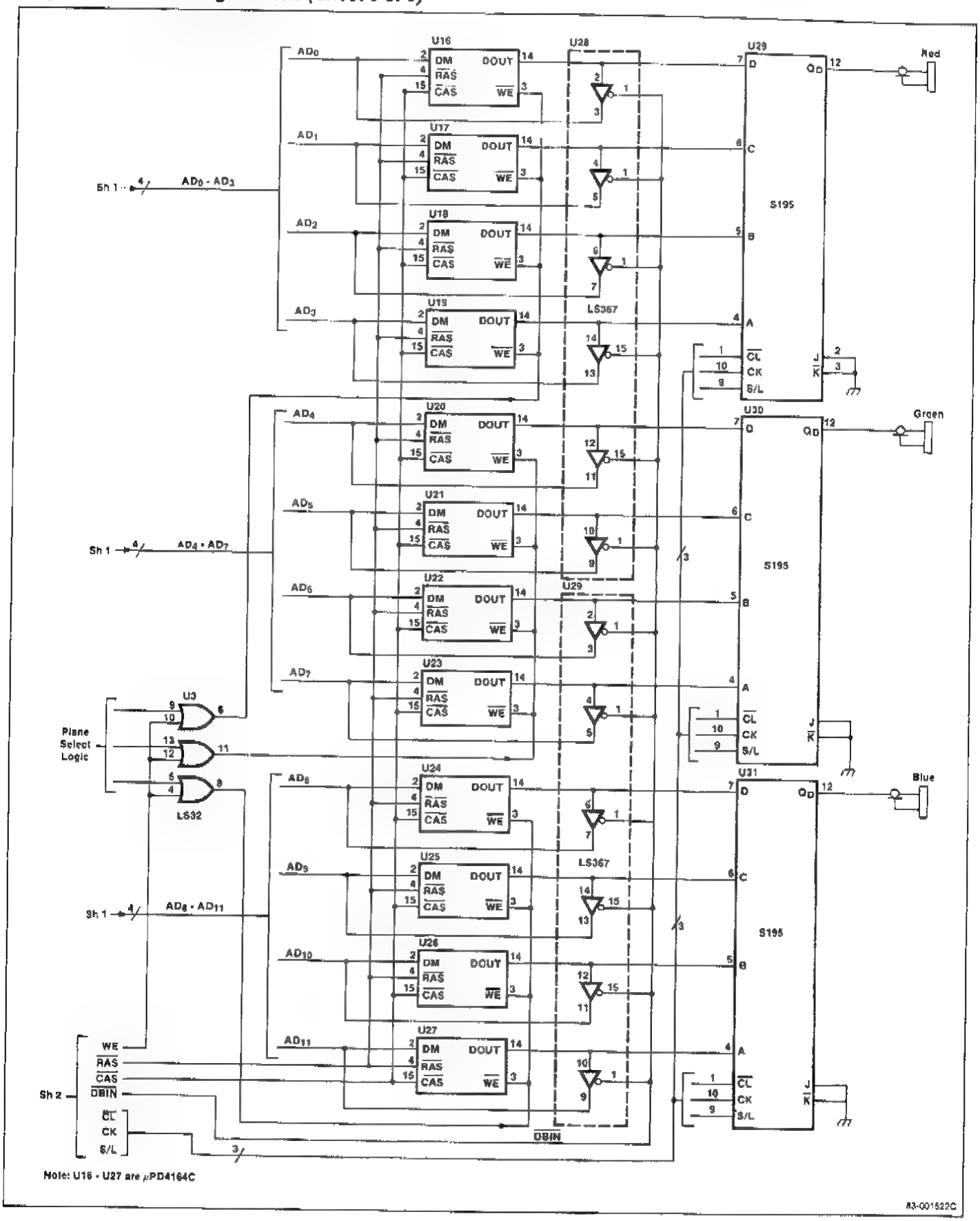
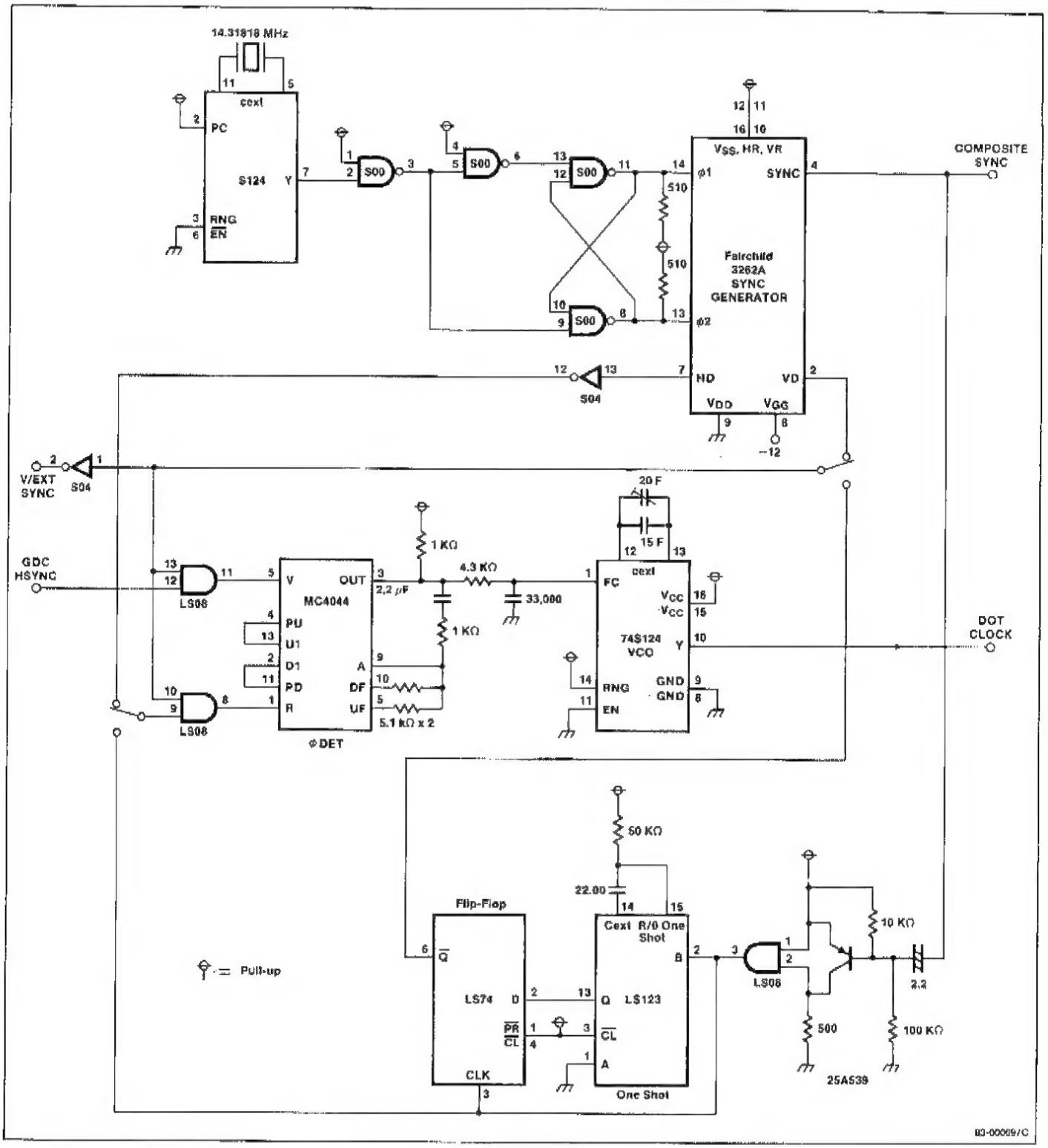


Figure F-1. Mask Register Bits (Sheet 3 of 3)



It is possible to use the GDC in a display apparatus to directly record character and graphic data created by the GDC on a video recorder without passing the data through a TV camera and overlaying the data to the video signal from a TV camera, a video disc, etc. This appendix describes a circuit (figure G-1) for synchronizing the GDC with signals conforming to the NTSC standard.

Figure G-1. NTSC Circuit



Synchronization to VSYNC [One Frame]

Vertical synchronization is established by setting the GDC to slave and connecting the VSYNC signal from a video apparatus such as a TV camera or video disc to the V/EXT SYNC pin.

When one frame period for the GDC is equal to one frame period of the video apparatus, this vertical synchronization is performed only once at the beginning of the GDC operation.

Synchronous aberrations after the vertical synchronization operation are corrected by horizontal synchronization with the PLL mentioned below.

Synchronization to HSYNC [One Line]

Horizontal synchronization is established with the following operations:

Locate the original oscillator that generates the dot-clock and provides the GDC with a divided dot-clock in the PLL.

Change the oscillating frequency by detecting the phase difference between two falling edges of the GDC's HSYNC and the HSYNC of the video apparatus. If the horizontal period (1H) of the GDC is equal to the horizontal period of the apparatus, do not change the oscillating frequency. The circuit in figure G-1 uses Fairchild's 3262A sync generator for a master sync signal generator. HD/VD signals are removed and only a composite sync signal is provided.

The 3262A generates sync signals that conform to the NTSC standard by supplying a 14.31818-MHz clock.

The phase-sensitive detector MC4044 detects the phase difference between the falling edges of the GDC's HSYNC and the HD signal from the 3262A and controls the voltage controlled oscillator (VCO), 74S124. The VCO lowers the frequency when advance is detected and raises the frequency when delay is detected.

Since the equalizing pulse generated during the VD signal is active, the input signals of the phase-sensitive detector are gated by the VD signal to remove the effect of the equalizing pulse.

A 25A539 separates the signals associated with the SYNC signal from the composite SYNC signal. The LS123 one-shot multivibrator cuts out nine horizontal periods after the beginning of VBLANK.

The SYNC signal conforming to NTSC standard is a master signal and the GDC is a slave. Vertical synchronization is roughly performed by the V/EXT SYNC function and strict horizontal synchronization is performed by the PLL. Therefore, the master VSYNC signal being input to V/EXT SYNC may be a rough signal, of which one period can be perceived by the GDC. It is not necessary to provide the master VSYNC signal to the slave GDC every vertical period.

Figure G-2 shows the commands that will allow the 7220 to be synched to an NTSC signal.

Figure G-2. Slave GDC Syncronization Commands and Parameters

Commands		Parameters
C	RESET	00
C	MASTER/SLAVE	6E (SLAVE)
C	SYNC	0E
P1		0A
P2		46
P3		05
P4		00
P5		08
P6		01
P7		F4
P8		24

Issue the START command after detecting the falling of VSYNC twice.

Notes:

REGIONAL SALES AND ENGINEERING SUPPORT OFFICES

NORTHEAST
Twenty Burlington Mall Road, Suite 449
Burlington, MA 01803
TEL 617-272-1774 TWX 710-348-6515

SOUTHEAST
Radice Corporate Center
600 Corporate Drive, Suite 412
Fort Lauderdale, FL 33334
TEL 305-776-0682 TWX 759839

MIDWEST
3025 West Salt Creek Lane, Suite 300
Arlington Heights, IL 60005
TEL 312-577-9090 TWX 910-687-1492

SOUTHCENTRAL
16475 Dallas Parkway, Suite 380
Dallas, TX 75248
TEL 214-931-0641 TWX 910-860-5284

SOUTHWEST
200 East Sandpointe, Building B Suite 460
Santa Ana, CA 92707
TEL 714-546-0501 TWX 759845

NORTHWEST
10080 North Wolfe Road, SW3 Suite 360
Cupertino, CA 95014
TEL 408-446-0650 TLX 595497

DISTRICT OFFICES

200 Broadhollow Road, Suite 302
Route 110
Melville, NY 11747
TEL 516-423-2500 TWX 510-224-6090

Beechwood Office Park
385 South Road
Poughkeepsie, NY 12601
TEL 914-452-4747 TWX 510-248-0066

200 Perinton Hills Office Park
Fairport, NY 14450
TEL 716-425-4590 TWX 510-100-8949

5720 Peachtree Parkway, Suite 120
Norcross, GA 30092
TEL 404-447-4409 TWX 910-997-0450

7257 Parkway Drive, Suite 109
Hanover, MD 21076
TEL 301-796-3944 TLX 759847

29200 Southfield Road, Suite 208
Southfield, MI 48076
TEL 313-559-4242 TWX 810-224-4625

Busch Corporate Center
6480 Busch Blvd., Suite 121
Columbus, OH 43229
TEL 614-436-1778 TWX 510-101-1771

8030 Cedar Avenue South, Suite 229
Bloomington, MN 55420
TEL 612-854-4443 TWX 910-997-0726

DISTRICT OFFICES [cont]

Echelon Building 2
9430 Research Boulevard, Suite 330
Austin, TX 78759
TEL 512-346-9280

6150 Canoga Avenue, Suite 112
Woodland Hills, CA 91367
TEL 818-716-1535 TWX 559210

Lincoln Center Building
10300 S.W. Greenburg Road, Suite 540
Portland, OR 97223
TEL 503-245-1600

5445 DTC Parkway, Suite 218
Englewood, CO 80111
TEL 303-694-0041 TWX 510-600-5666

NATICK TECHNOLOGY CENTER

One Natick Executive Park
Natick, MA 01760
TEL 617-655-8833 TWX 710-386-2110

NEC
NEC Electronics Inc.
CORPORATE HEADQUARTERS
401 Ellis Street
P.O. Box 7241
Mountain View, CA 94039
TEL 415-960-6000
TWX 910-379-6985

NECEL-000079-1285
STOCK NO. 500230